8-1-2018

A Bi-Encoder LSTM Model for Learning Unstructured Dialogs

Diwanshu Shekhar
University of Denver

A Bi-Encoder LSTM Model For Learning Unstructured Dialogs

A Thesis

Presented to

the Faculty of the Daniel Felix Ritchie School of Engineering and Computer Science

University of Denver

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Diwanshu Shekhar

August 2018

Advisor: Mohammad H. Mahoor, Ph.D.

Author: Diwanshu Shekhar
Title: A Bi-Encoder LSTM Model For Learning Unstructured Dialogs
Advisor: Mohammad H. Mahoor, Ph.D
Degree Date: August 2018

Abstract

Creating a data-driven model that is trained on a large dataset of unstructured dialogs is a crucial step in developing a Retrieval-based Chatbot systems. This thesis presents a Long Short Term Memory (LSTM) based Recurrent Neural Network architecture that learns unstructured multi-turn dialogs and provides implementation results on the task of selecting the best response from a collection of given responses. Ubuntu Dialog Corpus Version 2 (UDCv2) was used as the corpus for training. Ryan et al. (2015) explored learning models such as TF-IDF (Term Frequency-Inverse Document Frequency), Recurrent Neural Network (RNN) and a Dual Encoder (DE) based on Long Short Term Memory (LSTM) model suitable to learn from the Ubuntu Dialog Corpus Version 1 (UDCv1). We use this same architecture but on UDCv2 as a benchmark and introduce a new LSTM based architecture called the Bi-Encoder LSTM model (BE) that achieves 0.8%, 1.0% and 0.3% higher accuracy for Recall@1, Recall@2 and Recall@5 respectively than the DE model. In contrast to the DE model, the proposed BE model has separate encodings for utterances and responses. The BE model also has a different similarity measure for utterance and response matching than that of the benchmark model. We further explore the BE model by performing various experiments. We also show results on experiments performed by using several similarity functions, model hyper-parameters and word embeddings on the proposed architecture.

Acknowledgements

This thesis is dedicated to my parents for all the sacrifices, dedication and the hard-work they made to provide me with the best possible upbringing. The research for this thesis wouldn't have been possible without the advising of Dr. Mohammad Mahoor and Dr. Pooran Singh Negi. I am very grateful for their continued support. I would also like to thank faculty members - Dr. Matthew Rutherford, Dr. Rinku Dewry, Dr. Mike Goss and Dr. Scott Leutenegger for always encouraging me to do better and helping me navigate the path to becoming a good Computer Scientist. I would also like to offer special thanks to NVIDIA for providing GPU that provided computational parallelism needed in this research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Recently statistical techniques based on recurrent neural networks (RNN) have achieved remarkable successes in a variety of natural language processing tasks, leading to a great deal of commercial and academic interests in the field (Bengio et al., 2013; Cambria and White, 2014). Significant progress in the area of Machine Translation, Text Categorization, Spam Filtering, and Summarization have been made. Research in developing Dialog Systems or Conversational Agents - perhaps a desirable application of the future- have been growing rapidly. A Dialog System can communicate with human in text, speech or both and can be classified into - Task-oriented Systems and Chatbot Systems.

Task-oriented systems are designed for a particular task and set up to have short conversations from a single to at most 6 conversations. These systems interact with humans to get information to help complete the task. These include the digital assistants that are now on every cellphone or on home controllers and voice assistants such as Siri, Cortana, Alexa, Google Now/Home, etc.

Chatbot Systems, the area of this thesis, are systems that can carry on extended conversations with the goal of mimicking unstructured conversational or "chats" characteristic of human-human interaction. Lowe et al. (2015) explored learning models such as TF-IDF (Term Frequency-Inverse Document Frequency), Recurrent Neural Net-work (RNN) and a Dual Encoder (DE) based on Long Short Term Memory (LSTM) model suitable to learn from the Ubuntu Dialog Corpus Version 1 (UDCv1). We use this same architecture but on UDCv2 as a benchmark and introduce a new LSTM based architecture called the Bi-Encoder LSTM model (BE) that achieves 0.8%, 1.0% and 0.3% higher accuracy for Recall@1, Recall@2 and Recall@5 respectively than the DE model. In contrast to the DE model, the proposed BE model has separate encodings for utterances and responses. The BE model also has a different similarity measure for utterance and response matching than that of the benchmark model. We further explore the BE model by performing various experiments. We also show results on experiments performed by using several similarity functions, model hyper-parameters and word embeddings on the proposed architecture. This research has also been submitted for publication.

Chapter 1 describes a high level view of Machine Learning and Neural Network. Chapter 2 gives the background in related works where we describe the statistical language model and the neural network language model. We also describe the current state-of-the-art in vector representation of words and current research in chatbot systems. We describe the proposed BE model in Chapter 3 and conclude the paper in Chapter 4 with suggestions for potential future work.

For clarity, we establish a notation in this thesis wherein the type of the mathematical quantity involved will be denoted by its representation. Scalars are represented by lower case letters $i, j, k, \cdots ; \alpha, \beta, \gamma, \cdots$, vectors are represented via lower case bold letters $\boldsymbol{a, b, \cdots, e, \epsilon}, \cdots$ and matrices are represented by bold upper case letters $\boldsymbol{A, B}, \cdots, \boldsymbol{E}, \cdots$. Calligraphic letters $\mathcal{A}, \mathcal{T}, \cdots$ are used to represent sets of objects. We consistently follow similar convention for functions where f represents scalar valued functions, bold $\boldsymbol{f}$ represents vector valued functions and bold $\boldsymbol{A}_{i,.}$ represents the i^{th} row of the matrix A.

1.2 Machine Learning

Machine Learning is a method of approximating an unknown function given an input data and its corresponding output. In general, a machine learning process involves the following four steps:

1. Choosing a dataset
2. Choosing a model
3. Training the chosen model
4. Choosing an optimizer

1.2.1 Choosing a dataset

Datasets are an integral part of the field of machine learning. The availability of high quality training dataset is crucial from any machine learning task. The dataset can be labelled meaning, it can contain the desired output in addition to the input, or it can only contain the input in which case we call it unlabelled. Machine learning on the labelled dataset is known as supervised learning and on the unlabelled dataset is known as unsupervised learning. High-quality labeled training datasets for supervised and semi-supervised machine learning algorithms are usually difficult and expensive to produce because of the large amount of time needed to label the data. Although they do not need to be labeled, high-quality datasets for unsupervised learning can also be difficult and costly to produce.

As datasets come in myriad formats and can sometimes be difficult to use, there has been considerable work put into curating and standardizing the format of datasets to make them easier to use for machine learning research. Some of the common datasets can be found from the following repositories:

Kaggle Datasets: Online repository of machine learning datasets curated by the Kaggle community

NLTK Corpora: The Natural Language Toolkit, or NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It has a package called nltk.corpus which can be used through Python to access several datasets that have been cited in peer-reviewes publications. http://www.nltk.org/nltk_data/

PMLB: Penn Machine Learning Benchmarks or PNLB is a large, curated repository of benchmark datasets for evaluating supervised machine learning algorithms acccesible through Python (Geiger et al., 2012)

1.2.2 Choosing a model for training

The next step in the process of machine learning is choosing the right model or classifier for the machine learning task. There are several models to choose from - Naive Bayes, Support Vector Machines (SVM), Logistic Regression, Random Forest, K-means clustering, Neural Network, among others.

There is a general consensus among the machine learning community to start with a basic classifier and gradually increase the complexity of the model depending on the needs. For example: start with a Naive Bayer classifier for supervised learning or with a K-means clustering for unsupervised learning.

More often than not, the characteristics of the model also determines the process of model selection. For example:

- Random Forest: often very effective and can also perform regression
- K-means clustering: Simplest thing you can do, but often slow and requires lots of memory
- Neural Networks: slow to train, but very fast to run inference. Requires lots of memory. Can handle non-linearity
- SVM: Among the best with limited data, but not better against boosting or random trees when large data sets are available. Can handle non-linearity

One popular technique of choosing a model is parametric evaluation of each of the models. Let's say we have 4 models (Logistic regression, SVM, Random Forest and Neural Network) and 10 parameter combinations for each model. We simply have to evaluate the model for each model and parameter combination (4 * 10 = 40) and select the best model and parameters. Then we re-train with the best model and parameters on our training data and we have our final trained model.

1.2.3 Training a model

Training a model simply means searching for the optimum parameters of the model given a cost or loss function. The optimum parameters are those parameters that result in the lowest cost or loss. A cost function represent the price paid for inaccuracies of prediction in classification done by the machine learning task. The brief description of the commonly used machine cost functions are given below, assuming a simple linear model

$$y = Wx + b \tag{1.1}$$

where y is the predicted output of the model given the learned parameters W and b. The actual output from the dataset is denoted by y'. Some of the commonly used loss functions in training a model are described below:

- Squared Loss: While commonly used in regression, squared loss can be modified to be used in a classification problem. The square loss function tends to penalize outliers excessively leading to slower convergence rate than for the hinge loss.

$$\sum_{0}^{N} (y - y')^2 \tag{1.2}$$

- Hinge Loss: For binary classification problems, the network's predicted output is a single scalar y and the actual output y' is either $+1$ or -1. The classification rule is sign(y), and a classification is considered correct if $y' \cdot y > 0$, meaning that y and y' share the same sign. The hinge loss, also known as margin loss or SVM loss, is defined as:

$$\mathcal{X} = max(0, 1 - y' \cdot y) \tag{1.3}$$

  The loss is 0 when y' and y share the same sign and $y \geq 1$. Otherwise, the loss is linear. In other words, the binary hinge loss attempts to achieve a correct classification, with a margin of at least 1

  The hinge loss can also be extended to multi-class setting (Crammer and Singer, 2001). Let $\boldsymbol{y} = y_1 ... y_n$ be the predicted output vector, and $\boldsymbol{y'}$ be the one-hot vector for the correct output class. The classification rule is defined as selecting the class with the highest score: $prediction = argmax(\boldsymbol{y})$. The hingle loss for multi-class is given by:

$$\mathcal{X} = max(0, 1 - (y_t - y_k)) \tag{1.4}$$

where $t = argmax(\boldsymbol{y'})$ and $k = argmax(\boldsymbol{y})$

- Log Loss The log loss is a common variation of the hinge loss, which can be seen as a version of the hinge loss with an infinite margin (LeCun et al., 2006). It is given by:

$$\mathcal{X} = log(1 + exp^{-(y_t - y_k)}) \tag{1.5}$$

- Cross Entropy Loss: The cross entropy loss or the negative log likelihood loss is used when the probabilistic scores of the model for predicted classes are desired. It is given by the following:

$$\mathcal{X} = -\sum_{i=0}^{n-1} (p(i) \cdot log(i)) \tag{1.6}$$

  where n is the number of classes, and $p(i)$ is the probability score of the model for the class i.

- Ranking Loss: In some settings, we are not given supervision in term of labels, but rather as pairs of correct and incorrect items x and x', and our goal is to score correct items above incorrect ones. Such training situations arise when we have only positive examples, and generate negative examples by corrupting a positive example. A useful loss in such scenarios is the margin-based ranking loss, defined for a pair of correct and incorrect examples:

$$\mathcal{X} = max(0, s(x) - s(x')) \tag{1.7}$$

where s is the score assigned by the model.

1.2.4 Choosing an optimizer

The gradients of the loss function described in the previous section is calculated to update the parameters of the model until the optimum parameters are found.

The parameters, θ of the model can be updated by:

$$\theta = \theta - \alpha \cdot \Delta_\theta J(\theta) \tag{1.8}$$

where α is known as the learning rate of the mode and $J(\theta)$ is the mean loss of the model over the training dataset

This method of finding optimum parameters for the model is known as gradient descent. There are three variants of gradient descent:

- Batch gradient descent: The vanilla gradient descent algorithm described in (1.8) is called the batch gradient descent in which the gradients for the whole dataset is calculated to perform just one update to the parameters. Batch gradient descent can be very slow and is impractical for datasets that don't fit in memory

- Stochastic gradient descent: Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x_i and label y_i:

$$\theta = \theta - \alpha \cdot \Delta_\theta J(\theta; x_i; y_i) \tag{1.9}$$

SGD is therefore usually much faster.

- Mini-batch gradient decent: Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \alpha \cdot \Delta_{\theta} J(\theta; x_{i:i+n}; y_{i:i+n}) \tag{1.10}$$

However, mini-batch gradient descent does not guarantee good convergence and has a few challenges that need to be addressed:

- Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge

- Learning rate schedules try to adjust the learning rate during training by e.g. annealing (reducing the learning rate according to a pre-defined schedule or when the change in objective between epochs falls below a threshold). These schedules and thresholds, however, have to be defined in advance and are thus unable to adapt to a dataset's characteristics

- Additionally, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.

- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima and saddle points(points where one dimension slopes up and another slopes down) which are actually a bigger problem than the local minima (Dauphin et al., 2014).

To address these challenges, several gradient descent optimization algorithms have been developed such as:

- Adagrad
- RMSProp
- Adam

Since Adam Optimizer (Kingma and Ba, 2014) is used in the proposed Bi-Encoder model, we describe it in detail:

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past gradients m_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past squared gradients v_t. We compute the decaying averages of past and past squared gradients m_t and v_t respectively as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1.11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{1.12}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1)

They counteract these biases by computing bias-corrected first and second moment estimates:

$$m'_t = \frac{m_t}{1 - \beta_1^t} \tag{1.13}$$

$$v'_t = \frac{v_t}{1 - \beta_2^t} \tag{1.14}$$

They then use these to update the parameters, like in Adadelta and RMSprop, which yields the following Adam update rule:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{m'_t}{\sqrt{v'_t} + \epsilon} \tag{1.15}$$

1.3 Evaluating a machine learning model

One popular method of evaluating a trained model is called cross-validation. In cross-validation,we split our dataset into k non-overlapping subsets (folds), train a model using k-1 folds and predict its performance using the fold we leave out. We do this for each possible combination of folds. We estimate the mean performance of all folds and select the model that gives the best performance. Cross-validation is useful when the dataset is small and splitting the dataset into training, validation and the training set is not feasible.

If sufficient dataset is available, it is common to split the dataset into training, validation and the test dataset. During the training, the model is selected through evaluation on the validation dataset and after the final model is selected, we evaluate the model on the test dataset

1.4 Artificial Neural Network

Artificial Neural Network (ANN) was first introduced by McCulloch and Pitts (1943). They proposed a very simple model of the biological neuron, which later became known as an artificial neuron. It had one or more binary (on/off) inputs and one binary output. They showed that with a network of artificial neurons it was possible to compute any logical proposition (Figure 1.1 shows some logical propositions with ANN).

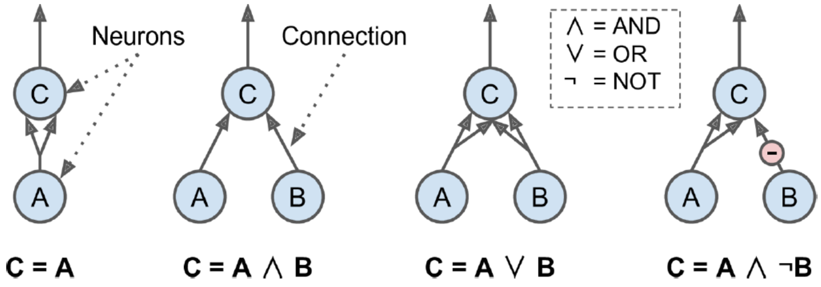


Figure 1.1: ANNs performing simple logical computations

Later in 1957, Rosenblatt (1957) came up with a simplest architecture of ANN where the inputs and the outputs were now numbers instead of the binaries and the neurons were the Linear Threshold Unit(LTU). He named his architecture Perceptron. Each connection from the input layer to an LTU will have some weight associated with it. The LTU computes a weighted sum of its inputs ($z = w_1x_1 + w_2x_2 + \ + w_nx_n = \boldsymbol{w}^T \cdot x$), then applies a step function to that sum and outputs the result: $h_w(x) = step(z) = step(\boldsymbol{w}^T \cdot x)$. Figure 1.2 shows a single layer of three LTUs with inputs coming from the input layer. The LTUs computes the weighted sum of these inputs and outputs 0 or 1 depending on the applied step function. This simple perceptron is able to classify instances into three different classes. Rosenblatt (1957), in his publication, also formulated a way to train his Perceptron ((1.16)).

$$w_{i,j}^{nextstep} = w_{i,j} + \alpha \cdot (y_j' - y_j) \tag{1.16}$$

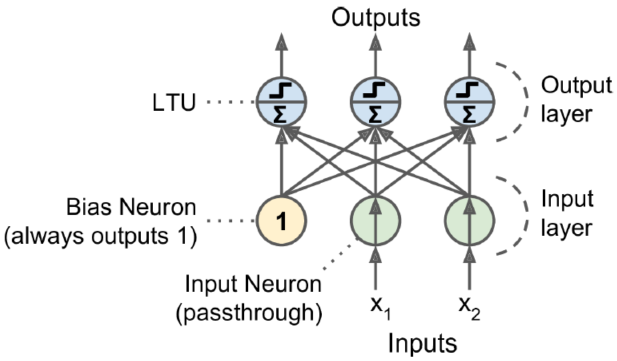


Figure 1.2: A Perceptron

where,

- $w_{i,j}$ = weight between the i^{th} input neuron and the j^{th} output neuron
- x_i is the i^{th} input value of the current training instance.
- y_j is the output of the j^{th} output neuron for the current training instance
- y'_j is the target output of the j^{th} output neuron for the current training instance
- α is the learning rate

However, the Perceptron described above has a serious limitation of not being able to solve XOR classification problem (this limitation was highlighted by Marvin Minsky and Seymour Papert in his book "Perceptrons: an introduction to computational geometry" published in 1969).

The XOR or the "Exclusive OR" classification problem is a problem of predicting an output of an XOR logic gates given two binary inputs.

As shown Figure 1.3, the XOR inputs are not linearly separable (which means no straight line can separate the outputs 1 and 0).

The processing unit of a single-layer Perceptron network is able to categorize a set of input patterns into two classes as the linear threshold function defines their linear separabil-

Input 1	**Input 2**	**Expected Output**
0	0	0
0	1	1
1	1	0
1	0	1

Table 1.1: XOR Logic Gates

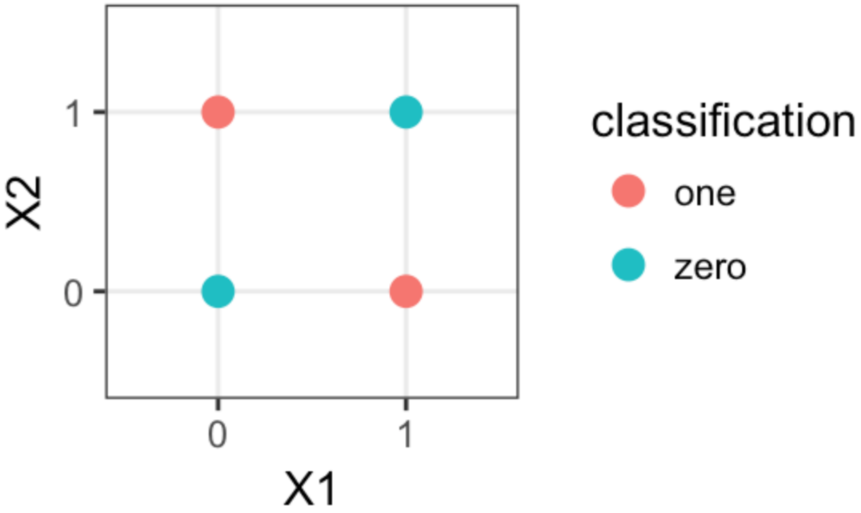
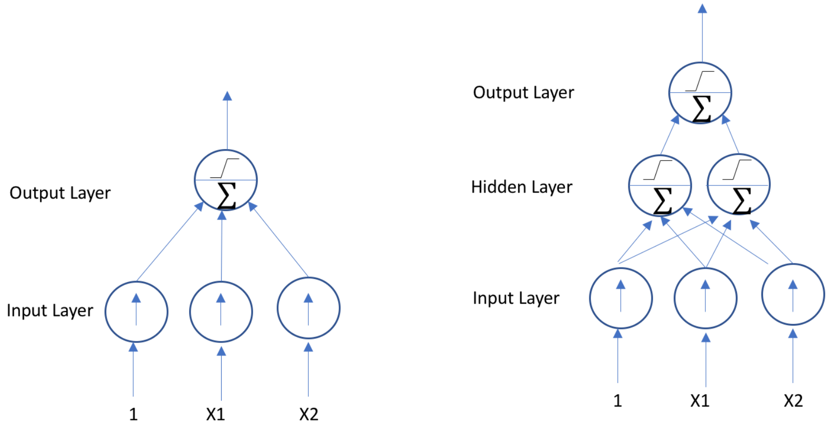


Figure 1.3: XOR Classification. X1 is Input 1 and X2 is Input 2

ity. Conversely, the two classes must be linearly separable in order for the single Perceptron network to function correctly. It turns out that by adding a layer to the single-layer Perceptron the XOR problem is resolved which gave rise to a new concept called "Multi-layer Perceptrons" (MLP) in ANNs (Figure 1.4)

An MLP is composed of one (passthrough) input layer, one or more layers of LTUs, called hidden layers, and one final layer of LTUs called the output layer. Every layer except the output layer includes a bias neuron and is fully connected to the next layer. When an ANN has two or more hidden layers, it is called a deep neural network (DNN).



(a) Single Layer Perceptron with XOR classification limitation (b) Multi-Layer Perceptron with XOR classification limitation resolved

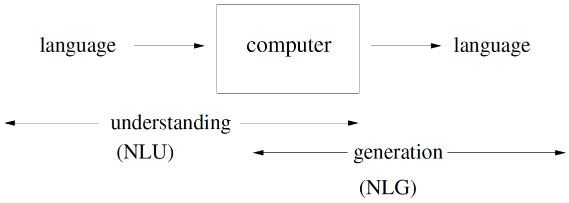
Figure 1.4: MLP that solved the XOR classification problem

For many years researchers struggled to find a way to train MLPs, without success. But in 1985, Rumelhart et al. (1985) introduced the back propagation training algorithm which is essentially the Gradient Descent algorithm described in the previous section.

1.5 Natural Language Processing

Natural Language Processing or NLP is an area of machine learning where computers use natural language as input and/or output. Figure 1.1 below illustrates a typical NLP system which consists of a subsystem called Natural Language Understanding (NLU) that feeds an input natural language to a computer and another system called Natural Language Generation (NLG) where a computer synthesizes natural language as a response to the input natural language.

Figure 1.5: Diagram of an NLP System



1.5.1 Why is NLP Hard?

As we have been using natural language since we were few years old, it may be surprising to hear that understanding human language is hard. The problem of understanding human language is so difficult that it is often said that NLP is "AI-complete". The main reason behind the complexity is the ambiguity behind the natural language. Darkin et al. (2002) explained various kinds of ambiguity in human language with the following example:
"At last, a computer that understands you like your mother"
The above sentence can have the following three different intents:

- It understands you as well as your mother understands you
- It understands (that) you like your mother
- It understands you as well as it understands your mother

To better understand, the ambiguity in human language can be classified into the following three categories:

- Syntactic Ambiguity
- Semantic Ambiguity
- Discourse Ambiguity

Syntactic ambiguity arises from different types part-of-speech parsing that form different parse trees for the same sentence. This is illustrated in Figure 1.6

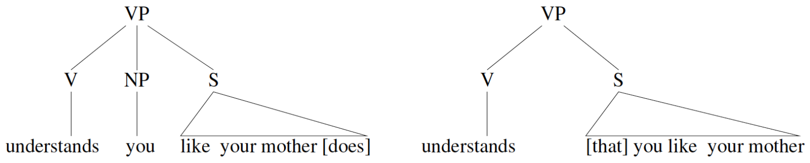


Figure 1.6: An Example of Syntactic Ambiguity

Semantic Ambiguity, on the other hand, arises from different dictionary meanings of the same word. For instance, there are two different meanings of the word "bank":

- a place where you deposit money
- nearby areas of a river or lake

Due to semantic ambiguity, the sentence "They put money in the bank" can be interpreted in the following two ways:

- They deposited money in some commercial bank such as Bank of America
- They buried money by some river

Another common type of ambiguity is called discourse ambiguity. Suppose we have the following text:

```
Alice says they've built a computer
that understands you like your mother.
She doesn't understand me at all.
```

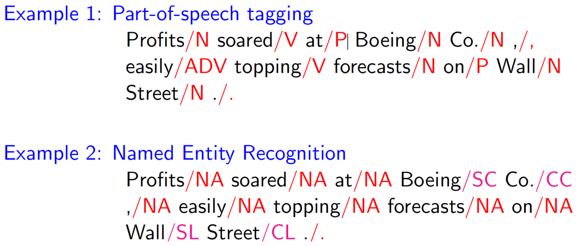
In the above text, it is unclear who "she" in the second sentence refers to. This is an instance of anaphora, where "she" co-referees to some other discourse entity.

1.5.2 Common Tasks in NLP

This section discusses the state of the art in solving some of the common problems in NLP.

Tagging

Converting a sequence of strings to some tagged sequence is the Tagging problem. This is illustrated in the two examples in Figure 1.7:



Example 1: Part-of-speech tagging
Profits/N soared/V at/P| Boeing/N Co./N ,/, easily/ADV topping/V forecasts/N on/P Wall/N Street/N ./.

Example 2: Named Entity Recognition
Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA topping/NA forecasts/NA on/NA Wall/SL Street/CL ./.

Figure 1.7: Examples of Tagging Problems

In the first example, words are tagged into a part-of-speech such as Noun (/N), Verb (/V) and so on while in the second example, the same words are tagged into named entities like Specific Company (/SC), Company Continuation (/SC), Specific Location (/SL) and Location Continuation (/LC). Words that are not named entities are tagged as /NA.

The tagging problems are supervised learning problems where a training sample consists of a sequence of words $x^i_1 ... x^i_n$ and a sequence of corresponding tags $y^i_1 ... y^i_n$ where i is the i^{th} sample in the training set. Our task would be to learn a function $f : X \rightarrow Y$ that maps sentences to tag sequences.

Parsing

The Parsing problem converts a sentence into a Parse Tree.

Parsing problems are also supervised learning problems where a training sample consists of parse tree of the syntactic structure of a sentence. For example: If a sentence in a training sample is "Boeing is located in Seattle" then the parse tree would be the following:

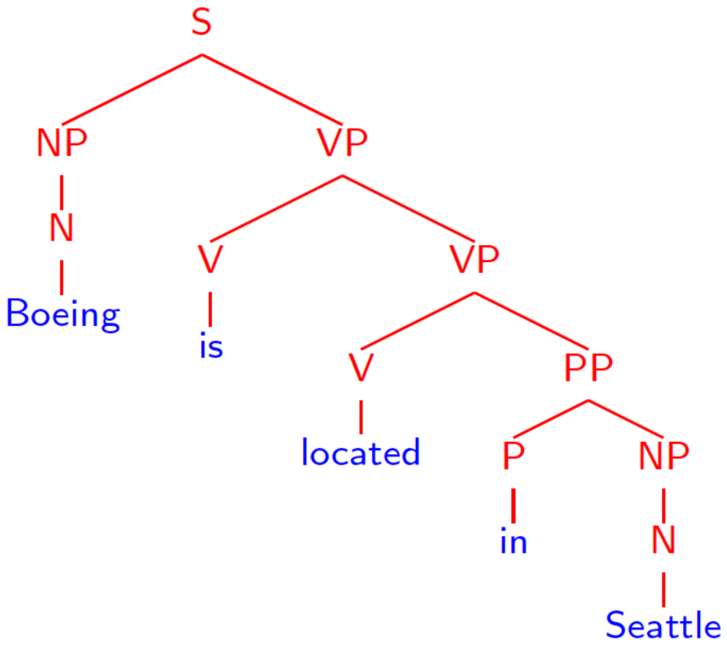


Figure 1.8: An Example of Parsing Problem

The words in a parse tree are at the leaf level nodes. The level just before the leaf level nodes are part-of-speech tags of the words and the rest of the internal nodes consist of the labels that identify some syntactic structure of the sentence. For instance, the group of words "in Seattle" is labeled as "Preposition Phrase(PP)"; "located in Seattle" is labeled as "Verb Phrase(VP)" and so on. Finally, "Boeing is located in Seattle" is labeled as Sentence(S). Parse trees give a useful grammatical relationship between the words. For example, whenever we see a parse tree of the form shown in the left hand side of Figure 1.9:

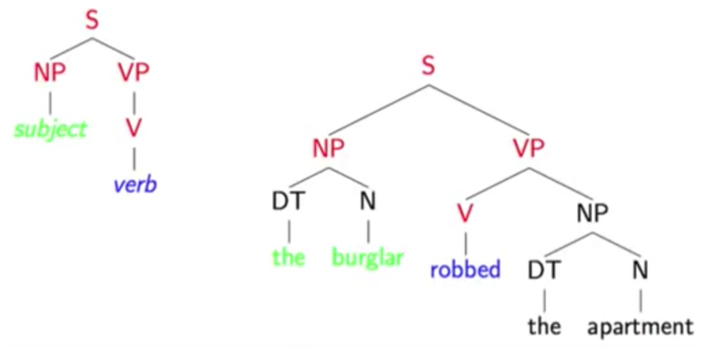
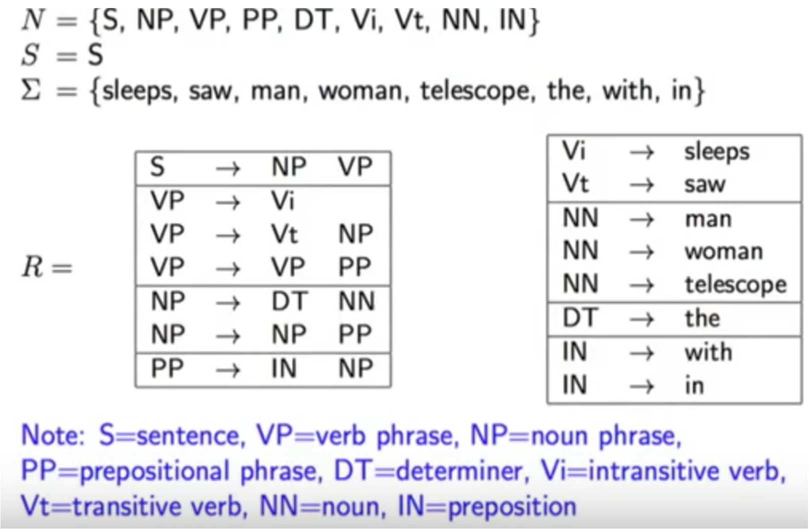


Figure 1.9: Relationship Between Words In A Parse Tree

We can say the words in green and purple have subject-verb relationship. Similarly, when we have a structure where V and NP are children of VP("robbed the apartment"), we can identify the relationship the words being as verb-object relationship. A very direct application of parse tree structure is in Machine Translation System. For instance in a Machine Translation System between English to Japanese, we know that English language follows more or less subject-verb-object structure and the Japanese language follows subject-object-verb structure. Knowing the inherent syntactic structure of the language, a parse tree of an English sentence can be converted to a parse tree of a Japanese sentence by simply rotating the nodes of the parse tree. Penn Treebank is a dataset of 50,000 sentences from Washington Street Journal that has been hand annotated by using the underlying syntactic theory.

Context-Free Grammars

First defined by Hopcroft (2013), the context free grammar, G is defined by four parameters (N, sum, R, S) where N a set of non-terminal symbols, $\sum$ a set of terminal symbols, R a set of rules of the form and S belongs to N is a designated start symbol. An example of a context free grammar in English is shown in the Figure 1.10:



N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, telescope, the, with, in}

$R =$

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
DT	→	the
IN	→	with
IN	→	in

Note: S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase, DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition

Figure 1.10: Relationship Between Words In A Parse Tree

The Left-Most Derivation Technique is used to derive a parse tree of a sentence using the context free grammar. We pick a start symbol, S and pick some rule from the set R and replace the left-most symbol with the derived symbols. We do this repetitively until all non-terminal symbols are replaced terminal symbols.

1.5.3 Applications of NLP

Machine Translation

Machine Translation (MT) is the the earliest research area in NLP. MT systems translate a word or sentence from one language to another. Google Translate is a good current example of an MT system.

Information Extraction

Given some text, the Information Extraction (IE) system generates a database of some keywords and their respective values. For example, given the following text:

```
10TH DEGREE is a full service advertising agency
specializing in direct and inter-active marketing.
Located in Irvine CA, 10TH DEGREE is looking for
an Assistant Account Manager to help manage
and coordinate interactive marketing initiatives
for a marquee automotive account. Experience in
online marketing, automotive and\or the advertising
field is a plus. Assistant Account Manager
Responsibilities Ensures smooth implementation of
programs and initiatives Helps manage the
delivery of projects and key client deliverables
: : : Compensation: $50,000-$80,000
Hiring Organization: 10TH DEGREE
```

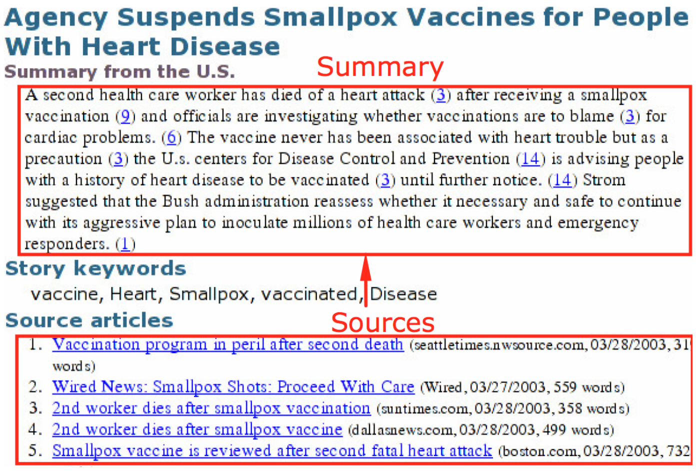
an IE will generate the following simple database of keys and values:

INDUSTRY	Advertising
POSITION	Assistant Account Manager
LOCATION	Irvine, CA
COMPANY	10th DEGREE
SALARY	$50,000 - $80,000

As illustrated in the above example, the goal of IE is to map a document to a structured database. The idea can be applied in performing complex searches (e.g., "Find me all the jobs in advertising paying at least $50,000 in Boston?") or performing statistical queries (e.g., "how has the number of jobs in accounting changed over the years?").

Text Summarization

Another application of Text Summarization (TS) is generating a summary of text from a group of texts from several sources. This is illustrated in Figure 1.11:



Agency Suspends Smallpox Vaccines for People With Heart Disease

Summary from the U.S.

A second health care worker has died of a heart attack (3) after receiving a smallpox vaccination (9) and officials are investigating whether vaccinations are to blame (3) for cardiac problems. (6) The vaccine never has been associated with heart trouble but as a precaution (3) the U.s. centers for Disease Control and Prevention (14) is advising people with a history of heart disease to be vaccinated (3) until further notice. (14) Strom suggested that the Bush administration reassess whether it necessary and safe to continue with its aggressive plan to inoculate millions of health care workers and emergency responders. (1)

Story keywords

vaccine, Heart, Smallpox, vaccinated, Disease

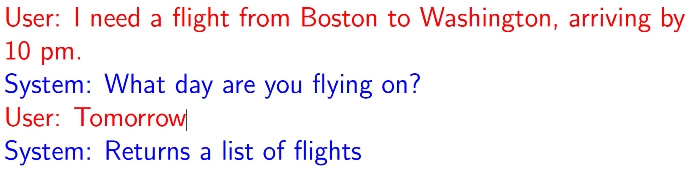
Source articles

1. Vaccination program in peril after second death (seattletimes.nwsource.com, 03/28/2003, 31 words)
2. Wired News: Smallpox Shots: Proceed With Care (Wired, 03/27/2003, 559 words)
3. 2nd worker dies after smallpox vaccination (suntimes.com, 03/28/2003, 358 words)
4. 2nd worker dies after smallpox vaccine (dallasnews.com, 03/28/2003, 499 words)
5. Smallpox vaccine is reviewed after second fatal heart attack (boston.com, 03/28/2003, 732

Figure 1.11: An Example Of Text Summarization

Dialogue Systems

In a Dialogue System, there is a conversation between a user and the system. The system processes the user's text to understand the intent and gives the answer back to the user. This is illustrated in Figure 1.12:



User: I need a flight from Boston to Washington, arriving by 10 pm.
System: What day are you flying on?
User: Tomorrow
System: Returns a list of flights

Figure 1.12: An Example Of Dialogue System

Chapter 2

Related Works

2.1 Statistical Language Modeling

Statistical NLP is the earliest approach to language modeling. It is a basic problem that is widely used in various applications of NLP.

A language model is defined as follows. First, we will define $\mathcal{V}$ to be the finite set of all words in the language. A sentence in the language is a sequence of words

$$x_1 x_2 ... x_n$$

where the integer n is such that $n \geq 1$, and $x_i \in \mathcal{V}$ for $i \in \{1...(n-1)\}$. We assume that x_n is a special symbol STOP which is not a member of $\mathcal{V}$. We will define $\mathcal{V}^\dagger$ to be the set of all sentences with the vocabulary $\mathcal{V}$.

Say,

$$\mathcal{V} = \{the, a, man, telescope, two, Beckham\}$$

then sentences in $\mathcal{V}^\dagger$ could be the following:

```
the STOP
```

```
a STOP
the man STOP
the man saw Beckham STOP
the man saw saw STOP
the man saw telescope STOP
the the the STOP
STOP
```

It is important to note that the set $\mathcal{V}^\dagger$ is infinite because sentences can be of any length. Given a set of training samples of example sentences, in $\mathcal{V}^\dagger$ where any sentence $x \in \mathcal{V}^\dagger$, a language model gives a probability distribution of x, $p\,(x)$ such that:

$$p(x) \geq 0$$

and

$$\sum_{x \in \mathcal{V}^\dagger} p(x) = 1$$

As mentioned earlier, language models are in used in broad range of applications, the most obvious one being in the area of speech recognition and machine translation. The distribution of probabilities of sentences given by the language models help predict the right sentence given a acoustic signal of a sentence. For example in speech recognition, if a language model is fed with the following two sentences:

```
recognize speech
wreck a nice beach
```

it will assign higher probability to the first sentence than the second one and will be able to accurately recognize the speech signal provided.

Although there are several elegant methods to train a language model which will be described in the later sections, a naive approach to training a language model would be to simply count the number of occurrences of a particular sentence in a training sample and defining the probability of that sentence:

$$p(x) = \frac{c(x)}{N} \tag{2.1}$$

where x is a sentence in the training sample and N is the total number of samples. However, this is poor model because it will assign a probability of zero to a new sentence not seen during the training of the model. Thus, this model fails to generalize to sentences not seen in the training samples.

In the following section, we will describe more robust language model such as the Trigram Model that are derived from Markov Processes.

2.1.1 Trigram Model

In this section, we derive an important class of language models called *trigram models* using the concept of Markov models. We first derive Markov models for fixed-length sequence of words and expand the concept to the variable-length sequence of words which is the case in practice.

Markov Models for Fixed-length Sequences

Consider a sequence of random variables, $X_1, X_2, ..., X_n$. Each random variable can take any word in a finite set $\mathcal{V}$. We make an assumption that all sequence of words is of fixed length n. We would like to model the joint probability of any sequence of words $x_1...x_n$

$$P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n)$$

Note that X_1 can take any word in the set $|\mathcal{V}|$, and so does X_2 and so on to X_n. Thus, there are $|\mathcal{V}|^n$ possible sequences of the form $x_1...x_n$. Clearly, it is not feasible for reasonable values of $|\mathcal{V}|$ and n to simply list all $|\mathcal{V}|^n$ probabilities. We use the concept of Markov processes to derive a more compact model. Markov processes can be of different orders - first order, second order, third order and so on. The order of up to three is usually sufficient in language modeling. In the first order Markov process, we make an assumption that $P(X_i = x_i)$ only depends on the previous word in the sequence $X_{i-1} = x_{i-1}$. We have the joint probability as:

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n) \\ = P(X_1 = x_1) &\prod_{i=2}^{n} P(X_i = x_i | X_1 = x_1, ..., X_{i-1} = x_{i-1}) \\ = P(X_1 = x_1) &\prod_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1}) \end{aligned} \tag{2.2}$$

Equation 2.2 shows the first order Markov process based derived from the exact form derived from the chain rule of conditional probability.

In a second order Markov process which will form the basis for trigram models, we make a slighter weaker assumption, namely that each word depends on the previous two words in the sequence:

$$\begin{aligned}
&= P(X_1 = x_1)P(X_2 = x_2)\prod_{i=3}^{n} P(X_i = x_i|X_1 = x_1, ..., X_{i-2} = x_{i-1}) \\
&= P(X_1 = x_1)P(X_2 = x_2)\prod_{i=3}^{n} P(X_i = x_i|X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})
\end{aligned} \tag{2.3}$$

It follows that the probability of an entire sequence is written as:

$$\begin{aligned}
&= P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n) \\
&= \prod_{i=1}^{n} P(X_i = x_i|X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})
\end{aligned} \tag{2.4}$$

For convenience, we will assume $x_0 = x_{-1} = *$ where * is a special start symbol in the sequence

Markov Models for Variable-length Sequences

In order to expand the fixed-length Markov model to a variable-length model, we assume that the n^{th} word in the sequence, X_n is always equal to a special symbol, the STOP symbol. This symbol can only appear at the end of the sequence and we make an additional assumption that this symbol is not part of the set $\mathcal{V}$. We have derived a second-order Markov process in Equation 2.4

where x_i can be a member of $\mathcal{V}$ or can be the STOP symbol. If we encounter the STOP symbol we finish the sequence. More formally, the process that generates sentences would be as follows:

1. Initialize i = 1 and $x_0 = x_{-1} = *$

2. Generate x_i from the distribution

$$P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

3. If $x_i = STOP$ then return the sequence $x_1 ... x_i$. Otherwise, set i = i + 1 and return to step 2

Thus, we now have a model that generated probability distribution of sequences that vary in length.

DERIVATION OF A TRIGRAM MODEL:

The trigram models are direct applications of Markov models. Under a second-order Markov model, the probability of any sentence $x_1 ... x_n$ is given by Equation 2.4 where we assume that $x_0 = x_{-1} = *$. For any i, x_{i-2}, x_{i-1} and x_i, we define a parameter q such that

$$P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1} = q(x_i | x_{i-2}, x_{i-1})$$

Our model then takes the form

$$P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = \prod_{i=1}^{n} q(x_i | x_{i-2}, x_{i-1}) \tag{2.5}$$

which is known as the trigram model. For any trigram (u, v, w), its parameter can be represented as

$$q(w | u, v)$$

For example, for the sentence

```
the dog barks STOP
```

we would have

$$p(the\;dog\;barks\;STOP) = q(the|*,*) \times q(dog|*,the) \\ \times q(barks|the,dog) \times q(STOP|dog,barks)$$

The key problem we are left with is to estimate the parameters of the model, namely

$$q(w|u,v)$$

ESTIMATING PARAMETERS OF A TRIGRAM MODEL:

Estimation of q parameters build on the idea of maximum-likelihood estimates. We define $c(u,v,w)$ to be the number of times that the trigram (u,v,w) is seen in the training sample. Similarly, we define c(u, v) to be the number of the times the bigram (u,v) is seen in the training sample. For any w, u, v we then define:

$$q(w|u,v) = \frac{c(u,v,w)}{c(u,v)} \tag{2.6}$$

In other words, the parameter value q of a trigram depends upon the ratio of the count of trigram and the count of bigram. This way of estimating parameters has two inherent downsides:

1. Many of the above estimates $q(w|u,v)$ will be zero due to the count in the numerator being 0

2. In cases where the denominator is equal to zero, the estimate is ill defined

These downsides are addressed in the Linear Interpolation Method described below:

Linear Interpolation Method

We define the trigram, bigram and unigram maximum-likelihood estimates as:

$$\begin{aligned} q_{ML}(w|u,v) &= \frac{c(u,v,w)}{c(u,v)} \\ q_{ML}(w|v) &= \frac{c(v,w)}{c(v)} \\ q_{ML}(w) &= \frac{c(w)}{c()} \end{aligned} \tag{2.7}$$

where $c(w)$ is the number of times word w is seen in the training sample, and $c()$ is the total number off words seen in the training sample. These maximum-likelihood estimates have their own strengths and weaknesses. While the trigram maximum-likelihood estimate captures the contextual sensitivity of the sentence, it has the sparse data problem as mentioned above. On the contrary, the unigram model do not have the sparse data issue but it doesn't capture the contextual sensitivity of the sentence. The bigram model falls between these two extremes. The Linear Interpolation uses all three estimates to capture contextual sensitivity of the sentence and address the issue of sparse data. The Linear Interpolation defines the trigram q parameter $q(w|u,v)$ as follows:

$$q(w|u,v) = \lambda_1 \times q_{ML}(w|u,v) + \lambda_2 \times q_{ML}(w|v) + \lambda_3 \times q_{ML}(w) \tag{2.8}$$

where the lambda parameters are the weights given to each maximum likelihood estimates such that:

$$\lambda_1, \lambda_2, \lambda_3 \geq 0$$

and

$$\lambda_1 + \lambda_2 + \lambda_3 = 0$$

There are various ways of estimating the λ parameters. A common one is using the method of optimization. We take a small portion out of the training sample as a validation set and define $c^{'}(u, v, w)$ to be the number of times the trigram is seen in the validation set. The likelihood of the validation set as a function of the $\lambda_1, \lambda_2, \lambda_3$ is

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{u,v,w} c^{'}(u, v, w) \log q(w|u, v) \tag{2.9}$$

We choose the λ values such that the function L is maximized. In practice, it is important to add an additional degree of freedom, by values of λ_1, λ_2 and λ_3 to vary depending on the bigram (u, w) that is being continued on.

2.2 Neural Network Language Model

A statistical language model learns the joint probability function of sequences of words in a language. However, this suffers from an intrinsic problem of curse of dimensionality: a word sequence on which the model will be tested is likely to be different from all the word sequence seen during the training. Although the aforementioned Trigram model solves this problem, a new approach to address this curse of dimensionality problem was proposed by Bengio et. al (2003) which is based on Neural Network model that learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations. The probabilistic neural network model solves two important problems that tri-gram models fails to address:

1. it is not taking into account contexts farther than two words

2. it is not taking into account the similarity between words

2.2.1 Recurrent Neural Network

Recurrent Neural Network (RNN), first introduced by Elman (1990) is a class of Artificial Neural Network (ANN) that can analyze a time-series data and predict something that is going to happen in the future. For example, given a time-series data of stock prices, it will be able to predict when to buy or sell. It can take sequences of words in a sentence or documents making it really useful in Natural Language Processing (NLP) applications.

RNNs are very much like feed-forward network except that the neurons also have as input their output leading to a feedback loop. This simple change makes them very powerful for learning from time series data. At each time-step $t-1$, the output of a neuron becomes the input to the same neuron in the next time-step t. Figure 2.1 shows a simple neuron unrolled through time. The output of the neuron at a particular time-step t is a function of its input at t and the output of the same neuron at the previous time-step $t-1$,

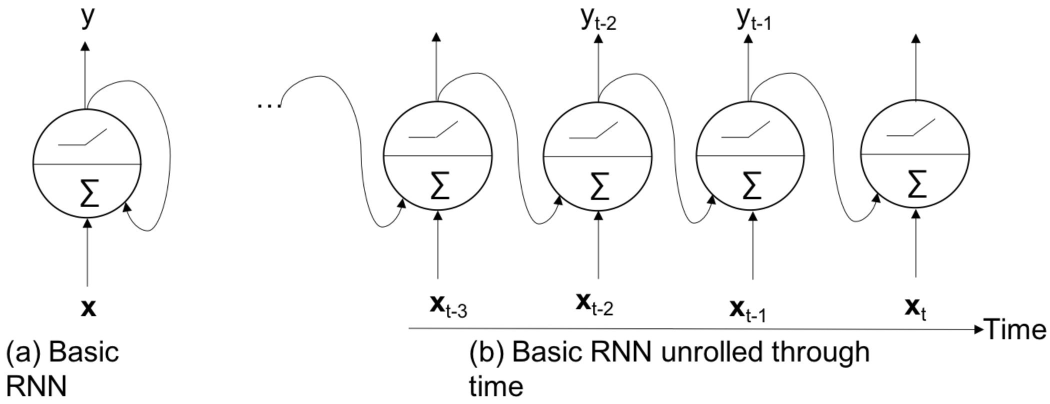


Figure 2.1: Basic RNN Cell

The output vector $\boldsymbol{y}_t$ of a single RNN neuron at a given time step t for a input vector $\boldsymbol{x}_t$ is given by:

$$\begin{aligned} \boldsymbol{h}_t &= \phi_H(\boldsymbol{W}_{xh} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{hh} \cdot \boldsymbol{h}_{t-1}) \\ \boldsymbol{y}_t &= \psi_O(\boldsymbol{W}_{hy} \cdot \boldsymbol{h}_t) \end{aligned} \tag{2.10}$$

where $\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_{hy}$ are shared connection weight matrices for various time steps and ψ and ϕ are activation functions. A natural extension of propagation algorithm called back propagation through time (BPTT) is used for training unrolled version of the neural network.

Training an RNN over long sequences require unrolling through many time-steps making it very deep neural network and like any deep neural networks, there is a problem of vanishing and exploding gradients that need to be addressed. The simplest and most common solution to this problem is to unroll the RNN only over a limited number of time steps during training. This is called truncated back propagation through time. But we will run into the risk of losing valuable time-dependent information and the model may not be able to learn long term patterns. Another problem with training RNNs on long sequences is that the memory of the first inputs gradually fades away due to several steps of transformations the data goes through while traversing the RNN. To address this issue, RNN cells are given a long term memory to hold valuable information from the past. One popular RNN cell with long term memory is called LSTM (Long Short Term Memory).

2.2.2 Long Short Term Memory based Recurrent Neural Network

The LSTM cell was first proposed by (Hochreiter and Schmidhuber, 1997) and several researchers, such as (Sak et al., 2014) and (Zaremba et al., 2014) have gradually made improvements. The basic architecture of an LSTM cell is shown in Figure 2.2:

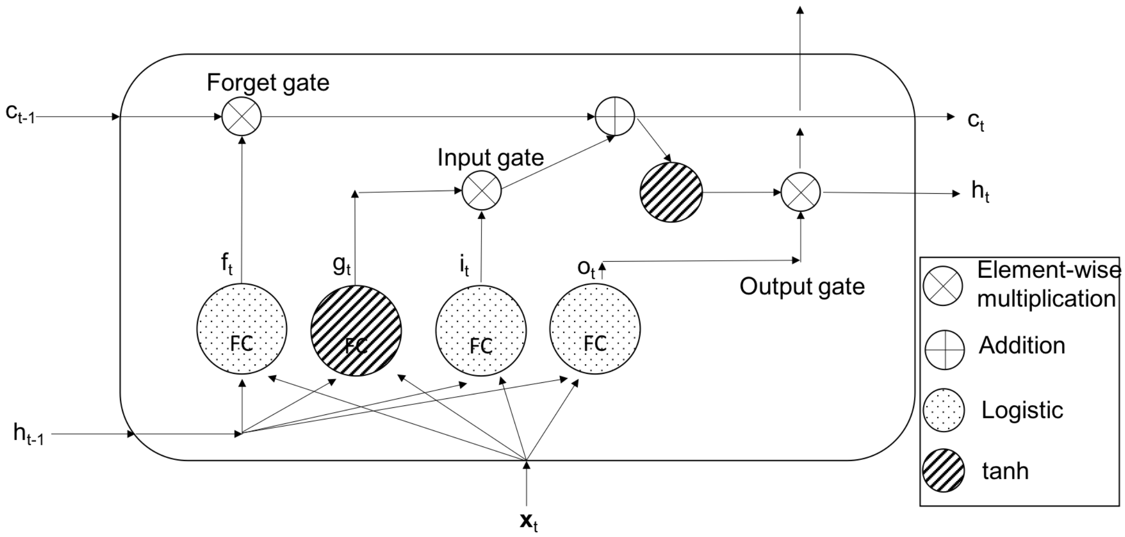


Figure 2.2: Architecture of an LSTM Cell

An LSTM cell is just like a regular RNN cell except that the state is now split into two vectors, $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$ where $\boldsymbol{h}_t$ stores the short term state and $\boldsymbol{c}_t$ stores the long-term state. The network learns what to store in the long-term state, what to throw away and what to read from it. These are accomplished by the use of gates - forget gate, input gate and the output gate. The forget gate controls which part of the long-term state should be thrown away. The input gate controls which part of the main output, $\boldsymbol{g}_t$ should be added to the long term state and the output gate controls which part of the long term state should be added to the cell's output. Equations below summarizes how to compute the cellâĂŹs long-term state $\boldsymbol{c}_t$, its short-term state $\boldsymbol{h}_t$, and its output $\boldsymbol{o}_t$ at each time step t for an input $\boldsymbol{x_t}$ in the sequence.

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{xi}^T \cdot \boldsymbol{x}_t + \boldsymbol{W}_{hi}^T \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{xf}^T \cdot \boldsymbol{x}_t + \boldsymbol{W}_{hf}^T \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_f \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{xo}^T \cdot \boldsymbol{x}_t + \boldsymbol{W}_{ho}^T \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_o \\
\boldsymbol{g}_t &= \tanh(\boldsymbol{W}_{xg}^T \cdot \boldsymbol{x}_t + \boldsymbol{W}_{hg}^T \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_g \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \boldsymbol{g}_t \\
\boldsymbol{y}_t &= \boldsymbol{h}_t = \boldsymbol{o}_t \otimes \tanh(\boldsymbol{c}_t)
\end{aligned} \tag{2.11}$$

where $\boldsymbol{W}_{xi}$, $\boldsymbol{W}_{xf}$, $\boldsymbol{W}_{xo}$,$\boldsymbol{W}_{xg}$ are the weight matrices of each of the four layers for their connection to the input vector $\boldsymbol{x}_t$ and $\boldsymbol{W}_{hi}$, $\boldsymbol{W}_{hf}$, $\boldsymbol{W}_{ho}$,$\boldsymbol{W}_{hg}$ are the weight matrices of each of the four layers for their connection to the previous short term state $\boldsymbol{h}_{t-1}$. The biases term for each of the four layers are $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, $\boldsymbol{b}_o$, $\boldsymbol{b}_g$. Due to there ability to retain long and short term memory, LSTM are able to achieve record results in machine translation (Sutskever et al., 2014) and automatic summarization (Nallapati et al., 2016).

2.3 Current State of Research on Chatbot Systems

The very first Chatbot system, ELIZA (Weizenbaum, 1966), was used for testing theories of psychological counseling. The Chatbot architecture was rule-based where a sentence is matched to a pattern and a response is provided based on the matched pattern. It was a simple algorithm but produced good results for the domain. Eliza's framework is still used today; modern Chatbot system tools like ALICE (Wallace, 2008) are based on updated versions of ELIZA's pattern/action architecture. A few years after ELIZA, another Chatbot with focus on clinical psychology, PARRY (Colby et al., 1971), was created to study schizophrenia. In addition to ELIZA like pattern matching, the PARRY system had a model of its own mental state of fear and anger. For example: certain topics of conversation might cause PARRY to become more angry or mistrustful. If PARRY's anger variable is high, it will choose from a set of âĂIJhostileâĂİ outputs.

However, these Chatbot systems were rule-based and needed domain expertise to handcraft rules in advance which made the design of these systems very time-consuming and expensive. To address this problem, corpora-based Chatbot systems have become the area of research since the last decade. Serban et al. (2015) did a survey of all available corpora for data-driven Chatbot systems. The availability of two large-scale corpora - Twitter corpora (Ritter et al., 2010) and the Ubuntu Dialog Corpus (Lowe et al., 2015) has opened several opportunities of research for corpora-based Chatbot system. Ritter et al. (2010) proposed the first unsupervised approach to the problem of modeling Dialog Acts. Just a year later using the same corpora, Ritter et al. (2011) showed that statistical machine translation could be used to generate a response given an utterance. This type of system is popularly now known as the Sequence to Sequence (seq2seq) Chatbot systems. It quickly became clear, however, that the task of response generation was too different from machine translation. In machine translation words or phrases in the source and target sentences tend to align well with each other; but in dialogs, a user utterance may share no words or phrases with a coherent response. Therefore, Shang et al. (2015) used transduction models for response generation instead of seq2seq models. A number of other modifications were made to the basic seq2seq model to adapt to the response generation task. Li et al. (2015) addresses the problem of the seq2seq models producing responses like "I'm OK" or "I don't know" that tend to end the conversation. Lowe et al. (2017) modifies the basic seq2seq model to resolve its inability to model the longer prior context by using hierarchical model. The basic seq2seq model focuses on generating single responses, and so don't tend to do a good job of continuously generating responses that cohere across multiple turns. This can be addressed by using reinforcement learning (Li et al., 2016), as well as techniques like adversarial networks (Li et al., 2017) that can select multiple responses that make the overall conversation more natural.

Another type of corpora based Chatbot systems that has been popular is the Information Retrieval (IR) based Chatbot systems. In the IR system, we find the most similar turn in the corpora that matches the user's utterance. Jafarpour et al. (2010) demonstrates the first attempt towards ranking a repository of responses to find the most suitable response. This was done by using filtering to reduce the size of the candidate responses, ranking based on feature selection, Multiple Adaptive Regression Trees (MART) and Transfer AdaBoost Algorithm. Later, Wang et al. (2013) created a dataset of short-conversations from a twitter-like micro-blogging platform and introduced a response model that used two stage approach - the first stage, similar to (Jafarpour et al., 2010) in idea but different in approach, used matching models to reduce the size of the candidate responses, and the second stage scored the reduced list of candidate responses with the help of a scoring model trained by RankSVM algorithm. Lowe et al. (2015) introduced an LSTM based RNN baseline response model to train on Ubuntu Dialog Corpus. Kadlec et al. (2015) improved the LSTM model introduced by (Lowe et al., 2015) by fine-tuning hyper-parameters and showed that an ensemble of LSTM, Bi-LSTM and CNN models performed better than the LSTM model.

The IR-based approach does not have to be confined to the corpora of dialogs, instead they can be generated from narrative texts. Isbell et al. (2000) generated responses by selecting sentences from a corpus that combined the Unabomber Manifesto by Theodore Kaczynski, articles on alien abduction, the scripts of âĂIJThe Big LebowskiâĂİ and âĂIJ-Planet of the ApesâĂİ. Chatbots that want to generate informative turns such as answers to user questions can use texts from Wikipedia to generate responses (Yan et al., 2016).

There is another type of Chatbot systems, at an early stage of an active research in NLP, called generative-based systems which does not require a repository of possible responses. Although, generative-based models seem to provide more flexibility to the question-answering agents, it is rather challenging owing to the much larger freedom the agent is allowed to respond with. (Kannan et al., 2016) demonstrated a hybrid system called Smart Reply that leverages both the Retrieval and Generative concepts. At the time of this research, the generative-based systems are not doing so well and most production systems are essentially retrieval-based such as - Cleverbot (Carpenter, 2011) and Microsoft's Little Bing system.

2.4 Dialogs Datasets

The dataset of dialogs or conversations can be broadly categorized into two types - Structured and Unstructured. Most tasks related to structured dialogs are focused on slot-filling as they can leverage the inherent structure in the dialogs. Voice-assistant systems such as Alexa and Google Now use structured dialogs and use pre-defined logical representation to answer user queries by filling slots in the responses. Appendix D shows an instance of DSTC-2 dataset. Structured data comes with a manual to understand how the dataset can be used for training and evaluation. Manuals for DSTC-2 and DSTC-3 dataset is available at http://camdial.org/~mh521/dstc/downloads/handbook.pdf. In contrast to structured dialogs, the unstructured dialogues have no a priori logical representation for the information exchanged during the conversation and can be used to train neural network based models.

The Switchboard dataset (Godfrey et al., 1992), and the Dialogue State Tracking Challenge (DSTC) datasets (Williams et al., 2013) have been used to train and validate dialogue management systems for interactive information retrieval. The problem is typically formalized as a slot filling task, where agents attempt to predict the goal of a user during the conversation. These datasets have been significant resources for structured dialogues, and have allowed major progress in this field, though they are quite small compared to datasets currently used for training neural architectures. Ritter et al. (2010) collected 1.3 million twitter conversations. Shang et al. (2015) used data from a similar Chinese website called Weibo. Later, Lowe et al. (2015) introduced the Ubuntu Dialogue Corpus version 1(UDCv1), a dataset containing almost 1 million multi-turn dialogues, with a total of over 7 million utterances and 100 million words. This provided a unique resource for research into building dialogue managers based on neural language models that can make use of large amounts of unlabeled data. The dataset has both the multi-turn property of conversations as in the Dialog State Tracking Challenge datasets, and the unstructured nature of interactions as in the microblog services such as Twitter

In this thesis, we primarily focus on unstructured dialogs which was pioneered by Ritter et al. (2011).

2.5 Word Vector Representation

The naive approach to representing a word is simply with a one-hot encoded vector in $R_{|V|}$ where V is a set of all vocabulary words. The meaning of a word is related to the distribution of words around it and representing words based on this idea is known as vector semantics.The shared intuition of vector space models of semantics is to model a word by embedding it into a vector space. For this reason the representation of a word as a vector is often called an embedding.

The earliest form of word vector representation was shown by Salton (1971) where he showed that a document can be represented by a vector of counts of words it contains and a word can be represented by a vector of its count in each document in a document base. This idea was presented in a Term Document Matrix (Table 2.1) where columns of a document represents the vector for the document and row of a target word (Term) represents its vector representation.

Term	**Doc 1**	**Doc 2**	**Doc 3**
I	0	0	3
like	2	1	1
hello	0	0	1
world	0	0	1

Table 2.1: Term Document Matrix

The Term Document matrix has been used in the classic Information Retrieval (IR) system where given a query, q (which is a considered a document), the best document matching the query could be found using some form of similarity measure of the vector representation of the query over all the documents, d (Equation 2.12).

$$\text{best matching document} = \operatorname*{argmax}_{d}\ sim(q, d) \tag{2.12}$$

In the Term Document Matrix, the size of the row vector of a word depends on the size of the document base. However, it is common to represent the Term Document Matrix as Term-Term in which the size of the matrix is the $|V|$ x $|V|$. Each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. It is most common, however, to use smaller contexts, generally a window around the word, for example of 3 words to the left and 3 words to the right, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a 3 word window around the row word. Table 2.2 shows a Term-Term matrix where the numbers co-occurrences of target words with the context words are shown for a window size of 5.

Terms	**...**	**the**	**is**	**of**	**which**	**in**	**...**
some	...	0	2	3	5	0	...
for	...	5	0	0	2	0	...
right	...	0	0	2	3	4	...

Table 2.2: Term Term Matrix

The aforementioned Term Document Matrix and the Term-Term Matrix are solely based on the counts. It turns out that this is not the best way to represent co-occurrences since rare words are more representative of a context or a document than the frequent words. A popular weighting scheme called Term Frequency Inverse Document Frequency (TF-IDF) can be used to modify the Term Document Matrix ((Luhn, 1957);(Sparck Jones, 1972)). Term Frequency (TF) is the number of times a target word appears in a document and the Inverse Document Frequency (IDF) is the fraction of the fraction of the total number of documents in the document base N, and the number of documents in which the target word appears df_i. IDF is given by the Equation 2.13 (log is used to dampen the number)

$$idf_{i,j} = log(\frac{N}{df_i}) \tag{2.13}$$

where, i, j is the i^th target word in the j^th document. The TF-IDF weight, $w_{i,j}$ is given by Equation 2.14

$$w_{i,j} = tf_{i,j} \cdot log(\frac{N}{df_i}) \tag{2.14}$$

The word vector representation techniques discussed so far are long (length of $|V|$) and sparse (most of the values are zero). Since it is beneficial to have short and dense (most values are non-zeroes) vectors in machine learning for the following two reasons -

- short vectors are easier to feed
- dense vectors generalize lot better than the sparse vectors

Several methods have been developed to represent word vectors as short and dense. Deerwester et al. (1990) devised a method called Latent Semantic Analysis (LSA) that leverages the concept of Singular Value Decomposition (SVD) to reduce a long rectangular $|V|$ by c document or context matrix to a $|V|$ by k matrix where k is much shorter than c, usually set to 300. Later, Schütze (1992) showed that LSA can not only be applied to Term-Document Matrix, but also to the Term-Term matrix.

Another approach to generating short and dense word vectors comes from a different family of methods is based on prediction from Neural Network based language models. Mikolov et al. (2013) proposed skip-gram and Continuous Bag of Words (CBOW). Because the software package that implements these two methods is called word2vec, this method is also popularly known as word2vec. The skip-gram and the CBOW are mirror image of each other. The skip-gram model computes the words, w^{t-1}, w^{t-2}, w^{t+1}, w^{t+2} in a context size of L given the current word and the CBOW model computes the current word, w^tgiven the context words w^{t-1}, w^{t-2}, w^{t+1}, w^{t+2} (here, $L = 4$).

While methods like LSA efficiently leverage statistical information, they do relatively poorly on the word analogy task, indicating a sub-optimal vector space structure. Methods like skip-gram may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts. To address this issue, Pennington et al. (2014) presented a new method where they make use of the global word-word occurrence counts and formulate a weighted least squares model that could be optimized without the need for an on-line training unlike the skipgram or the CBOW model. The word vectors generated through this method are called GloVe vectors. They show that GloVe vectors outperform other current methods on several word similarity tasks, and also on a common named entity recognition (NER) benchmark. In this thesis, we use GloVe vectors for word embeddings.

Chapter 3

Proposed Architecture: Bi-Encoder LSTM

3.1 Architecture

The proposed BE model architecture in Figure 3.1 is motivated by the typical setup of conversation between two persons. Each person has to encode long and short term conversation contexts to best respond to a spoken sentence (an utterance or context).

As a natural design choice, in the BE model one LSTM cell (colored in grey) learns encoding of an utterance (or questions or contexts) while the other LSTM cell (colored in white) learn encoding of a response (or answers or responses). A sequence of GloVe embedding vectors (Pennington et al., 2014) of an utterance are fed into the upper LSTM while the sequence of embedding vectors of a response are fed into the lower LSTM cell. Vectors representing the final states $\boldsymbol{h}_t \in \mathbb{R}^s$ of the upper and lower LSTM cells are used for final representations of the utterance and the response as $\boldsymbol{u}_e$, $\boldsymbol{r}_e$ respectively. To drive learning of $\boldsymbol{u}_e$ and $\boldsymbol{r}_e$, we measure their similarity in the hidden vector space using dot product i.e

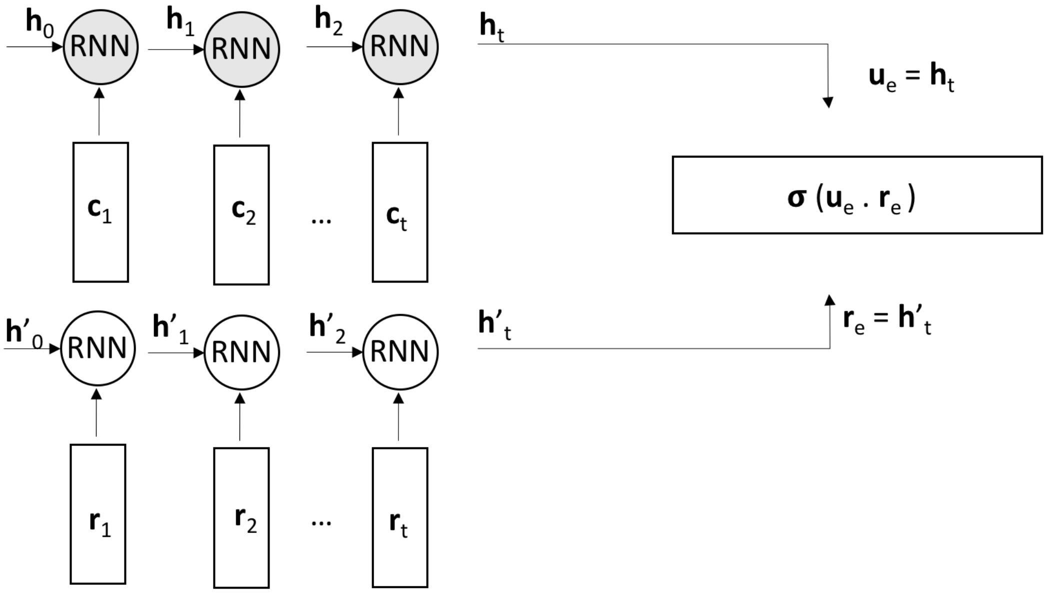


Figure 3.1: Bi Encoder LSTM Architecture. RNNs are colored in grey and white to show two different LSTM networks

$$\text{sim}(\boldsymbol{u}_e, \boldsymbol{r}_e) = \langle \boldsymbol{u}_e^T, \boldsymbol{r}_e \rangle \tag{3.1}$$

The training of the model via BPTT is done by minimizing the binary cross entropy $\mathcal{X}(q,\, p)$ between the learned probability p and the ground truth paring probability $q = \{0,\, 1\}$, where 1 denotes $\boldsymbol{u_e}, \boldsymbol{r_e}$ are genuinely paired and 0 denotes otherwise. Using similarity in Eq. (3.1), p is calculated as:

$$p = \frac{1}{1 + e^{(\text{sim}(\boldsymbol{u}_e, \boldsymbol{r}_e) + \boldsymbol{b})}} \tag{3.2}$$

In eq. (3.2), b is a scalar free parameter bias that is learned by the model. The aforementioned cross entropy loss is given by:

$$\mathcal{X}(q,\, p) = -q \cdot log(p) - (1-q)log(1-p) \tag{3.3}$$

The model is trained using the Adam Optimizer (Kingma and Ba, 2014) with a learning rate of 0.001 by minimizing the loss function in Eq. (3.3).

Figure 3.1 shows an LSTM cell that has been unrolled through time. c_1, c_2, ..., c_t represents the context word at time $t-1$, $t-2$, ..., t respectively. Similarly, r_1, r_2, ..., r_t represents the response word at time $t-1$, $t-2$, ..., t respectively. The context (upper) part of the architecture can be unrolled through the maximum time of 160, and the response (lower) part of the architecture can be unrolled through the maximum time of 80. This was done to address the fact that the average length of responses are almost half of the average length of contexts in the training dataset. The words in the context or the response are fed into the LSTM cell one by one from left to right up to the maximum unrolled time allowed for the LSTM cell. Because we use RNN, any word at c_t will learn the semantic of the word based on the context from the all previous words, c_{t-1}, c_{t-2} and so on. The same applies true for the words in the responses. Thus, when a full sentence is fed into an LSTM cell, the semantic of the sentence is represented by the final state of the LSTM cell. In Figure 3.1, h_t represents the semantic vector of a context sentence and h_t' represents the semantic vector of a response sentence. We compare how best the encoded response matches with the encoded utterance(context) by using the dot product. The model optimizes its weights (in other words, learns its optimum parameters) by minimizing the divergence between the encoded context and the response vectors.

Appendix A shows the hyper-parameters used in the BE model.Figure 3.2 shows the main portion of the computational graph of the BE model defined in Tensorflow v1.7.

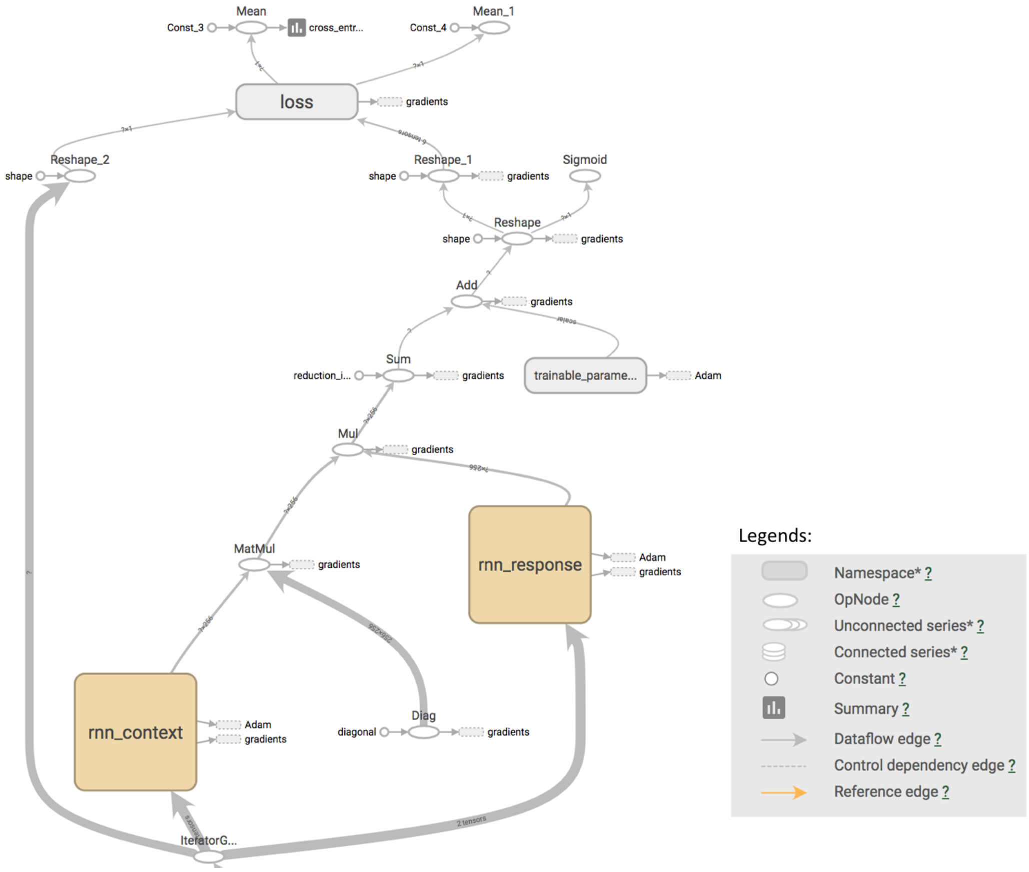


Figure 3.2: Main portion of the computational graph of the BE model defined in Tensorflow v1.7

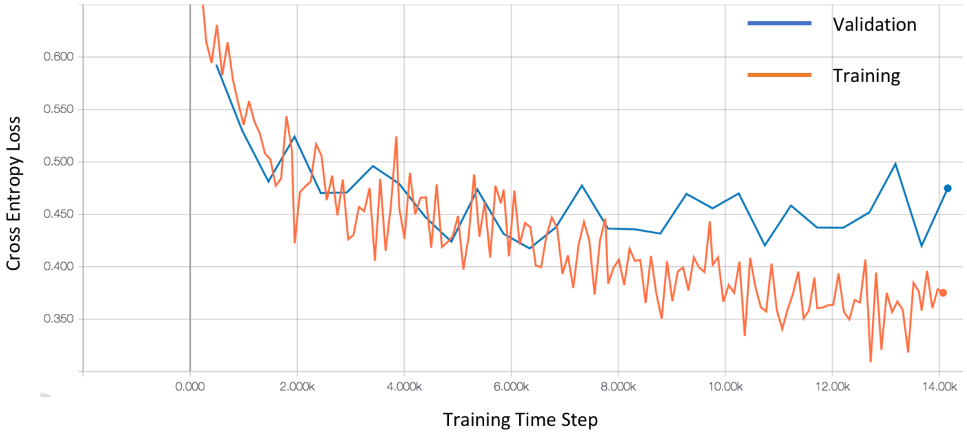


Figure 3.3: Cross entropy loss curve of the BE model from start to the end of the training. Figure shows validation and the training loss. The best model was obtained at 6,834 steps which is approximately 6 epochs

The BE model takes about 2 hours to train and we use early stopping strategy to prevent the model from over-fitting. Figure 3.3 shows the cross entropy loss curve of the BE model from start to the end of the training.

Figure 3.4 shows the Cumulative Match Characteristic (CMC) curve that shows the true positive identification rate of the BE model for Recall@k for k $\in \{1, 2, ..., 10\}$.

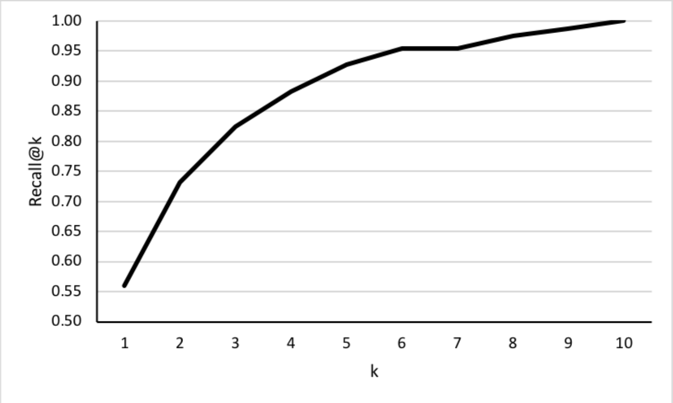


Figure 3.4: CMC Curve of the BE Model

In this subsequent section, we look at various experiments that helped us decide to select the best similarity function, hyper-parameters and word embedding for the BE model. We also show performance of the BE model in comparison to the DE model.

3.2 Experiments and Results

All our models were implemented in Tensorflow v1.7 and trained using a GeForce GTX 1080 Ti NVIDIA GPU. We used the same training data, UDC as (Lowe et al., 2015) but its second version (UDCv2). Models are trained on 1 million pairs of utterances and responses on the training set and evaluated against a test set. We fine-tune the model with hyper-parameters, determine the optimum similarity function and word embedding using the validation dataset.

For evaluation and model selection, we present our model with 10 response candidates, consisting of one right response and the rest nine incorrect responses. This set of 10 response candidates per context is provided in the validation and test set in UDCv2 (more details in Section 3.2.1). The model ranks these responses and its ranking is considered correct if the correct response is among the first k candidates. This quantity is denoted as Recall@k. Specifically, we report mean values of Recall@1, Recall@2 and Recall@5.

For benchmarking, we use the DE model in (Lowe et al., 2015) and the results of the DE model on UDCv2 as published in
https://github.com/rkadlec/ubuntu-ranking-dataset-creator. In contrast to the BE model, the DE model has one LSTM cell that encodes both the utterance and the response. The encoding for the utterance, $\boldsymbol{u}_e$ is multiplied with a trainable matrix $\boldsymbol{M}$ whose result is compared with the encoding for response, $\boldsymbol{r}_e$ by a dot product (Figure 3.5).

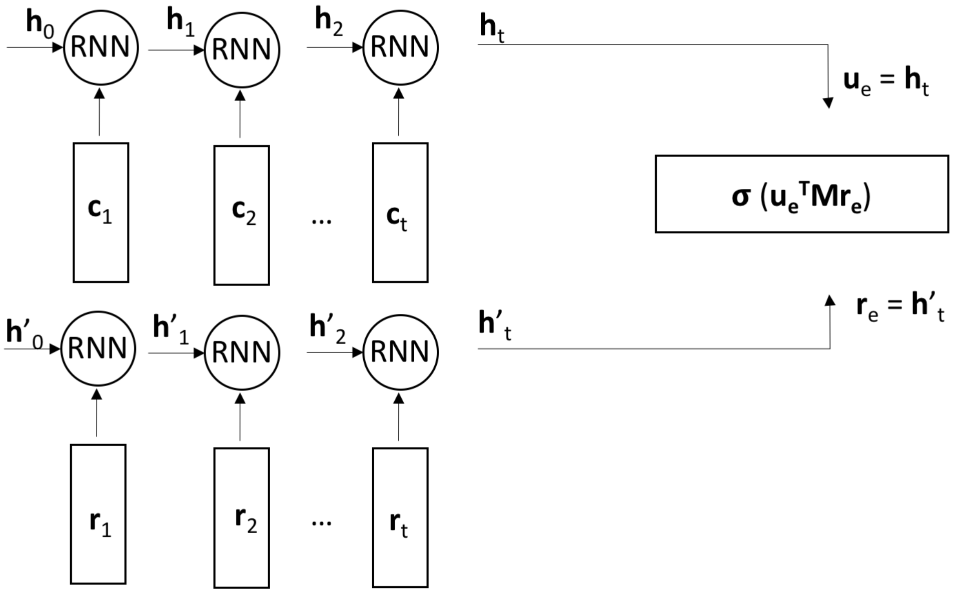


Figure 3.5: DE Model. All RNNs are colored in white to show the same LSTM network has been fed first by the utterance and then by the response

We also reproduced the DE model for comparison and we refer it as the DER model. Note that the DE model was originally modeled and trained in Theano. We speculate that accuracy values of the reproduced DE model not exactly matching with accuracy values of the benchmark model from Lowe et al. (2015) is attributed to the differences in hyperparameters.

3.2.1 Data

The Ubuntu Dialog Corpus (UDC) is the largest freely available multi-turn dialog corpus (Lowe et al., 2015). It was constructed from the Ubuntu chat logs - a collection of logs from Ubuntu-related chat rooms on the Freenode IRC network. Although multiple users can talk at the same time in the chat room, the logs were pre-processed using heuristics to create two-person conversations. The resulting corpus consists of almost one million two-person conversations, where a user seeks help with his/her Ubuntu-related problems

	Utterance Length	**Response Length**
mean	86.33	17.24
std	74.92	16.42
min	5	1
30%	42	8
60%	77	16
90%	179	36
max	1879	653

Table 3.1: Statistical Distribution of the utterances and responses lengths. Table shows mean, standard deviation, the minimum, percentiles and the maximum values of lengths

(the average length of a dialog is 8 turns, with a minimum of 3 turns). Because of its size, the corpus is well-suited for deep learning in the context of dialogue systems.

UDCv2 released in 2017 made several significant updates to its predecesor (https://github.com/rkadlec/ubuntu-ranking-dataset-creator). To summarize - UDCv2 is separated into training, validation and test set; the sampling procedure for the context length in the validation and test set is changed from an inverse distribution to a uniform distribution; the tokenization and entity replacement procedure was removed; differentiation between the end of an utterance (__eou__) and end of turn (__eot__) has been added; a bug that caused the distribution of false responses in the test and validation sets to be different from the true responses was fixed.

The training set consists of labelled 1 million pairs of utterances and responses. It has equal distribution of true context-response pairs labeled as 1 versus the context-distraction pairs labeled as 0. Keeping all the words that occur at least 5 times, the training set has a vocabulary of 91,620. The average utterance is 86 words long and the average response is 17 words long. Table 3.1 shows more details on the statistical distribution of the lengths of the utterances and responses.

The validation dataset consists of 19,560 examples where each example consists of a context and 10 responses where the first response is always the true response. The test

Context	Response	Label
you just click userprefer and creat one ... __eou__ __eot__ i ca n't access the wiki at all - it throw http auth at me __eou__ __eot__	unless nat get distract by someth shinier	1
i think that tutori be outdat - veri first instruct fail __eou__ and it differ slight to what ubotu have to say __eou__	i think that tutori be outdat - veri first instruct fail __eou__ and it differ slight to what ubotu have to say __eou__	0

Table 3.2: Examples from the training dataset of UDCv2 showing both the correct (1) and incorrect response (0) labels

dataset, structured the same as the validation dataset, consists of 18920 examples. The correct response is the actual next utterance in the dialogue and a false response is randomly sampled utterance from elsewhere within a set of dialogues in UDC that has been set aside for creation of validation and test set (Lowe et al., 2015). The words of the UDCv2 are stemmed (strip suffixes from the end of the word), and lemmatized (normalize words that have the same root, despite their surface differences).

3.2.2 Effect of similarity measures

In the BE model, we used dot product similarity between the encoded utterance $\boldsymbol{u}_t$ and response $\boldsymbol{r}_e$ and we wanted to see how other types of similarity measures effect the performance of the model. The description of these similarity measures are given in the subsequent sections.

Cosine Similarity

Instead of taking the dot product of $\boldsymbol{u}_e$ and $\boldsymbol{r}_e$, we take the dot product of their unit vectors. This is shown in the following equation:

$$\mathrm{sim}(\boldsymbol{u}_e, \boldsymbol{r}_e) = \frac{\boldsymbol{u}_e^T \cdot \boldsymbol{r}_e}{|\boldsymbol{u}_e|\,|\boldsymbol{r}_e|} \tag{3.4}$$

Polynomial Similarity

In machine learning, the polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models. Although the Radial Basis Function (RBF) kernel is more popular in SVM classification than the polynomial kernel, Goldberg and Elhadad (2008) showed that polynomial kernel gives better result than the RBF-Kernel for NLP applications:

For degree$-d$ polynomials, the polynomial kernel is defined as:

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \cdot \boldsymbol{y} + c)^d \tag{3.5}$$

where x and y are vectors in the input space, i.e. vectors of features computed from training or test samples and $c \geq 0$ is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. When $c = 0$, the kernel is called homogeneous.

In this experiment, we used the polynomial kernel from 0^{th} to the 3^{rd} degree for the similarity measure. The following equation gives the similarity function:

$$\mathrm{sim}(\boldsymbol{u}_e, \boldsymbol{r}_e) = \sum_{d=0}^{3} (\boldsymbol{u}_e^T \cdot \boldsymbol{r}_e)^d \tag{3.6}$$

3.2.3 Effect of using all hidden states

In this experiment, we used all the hidden states of the LSTM and not just the final states to encode the utterance and response. The encoding for the utterance is given by:

$$\boldsymbol{u_e} = \sum_{t=1}^{T} \frac{t^2}{T^2} \cdot \boldsymbol{h}_t \tag{3.7}$$

where T is the maximum context length of the utterance

Similarly, the encoding for the response is given by:

$$\boldsymbol{r_e} = \sum_{t=1}^{T} \frac{t^2}{T^2} \cdot \boldsymbol{h'_t} \tag{3.8}$$

We keep the similarity function as in the original BE model as shown in Eq. (3.1).

3.2.4 Deep LSTM

In this experiment, we added two more layers to the shallow LSTM BE model and looked at the result. We keep the similarity function as in the original BE model as shown in Eq. (3.1).

3.2.5 Bi-directional LSTM

In this experiment, we are interested in looking at the effect of bi-directional LSTM to the BE model

3.2.6 Results and Discussion

Table 3.3 compares the performance of various retrieval based models on UDC dataset. Compared to the benchmark DE model, the proposed BE model achieves 0.8%, 1.0% and

Model Id	Description	Recall@1	Recall@2	Recall@5
DE	Dual Encoder LSTM (Benchmark)	55.2	72.09	92.43
DER	Dual Encoder LSTM Reproduced	52.6	70.09	91.51
BE	Bi-Encoder LSTM (Proposed)	**56.0**	**73.15**	**92.7**

Table 3.3: Comparison of top-k % accuracy on UDCv2 on the test set

Model Id	Description	Recall@1	Recall@2	Recall@5
BE-19	BE with Cosine similarity	43.09	61.99	86.97
BE-20	BE with Polynomial Similarity	55.11	71.64	92.17
BE-21	BE using all hidden states	54.7	71.54	91.63
BE-22	BE with deep LSTM model	53.8	71.6	92.4
BE-23	BE with bi-directional LSTM model	55.86	72.50	92.69
BE	BE with Dot Similarity	**56.88**	**73.24**	**92.86**

Table 3.4: Results of different similarity measures used on the BE model using the validation set

0.3% higher accuracy for Recall@1, Recall@2 and Recall@5 respectively. Note that compared to the reproduced DE model, the BE model does better than when it is compared to the benchmark model.

Table 3.4 shows the results of various experiments we performed on the BE model.

Upon making a comparison between the proposed BE and the benchmark DE model, we try to see how the BE model with regular RNN (BE-RNN) compares with the DE model with regular RNN (DE-RNN). This comparison is shown in Table 3.5. The results of the DE-RNN model on UDCv2 are as published in https://github.com/rkadlec/ubuntu-ranking-dataset-creator. The DE-RNNR is the DE-RNN model that was reproduced. We keep the hyper-parameters used in the DE-RNN the same as the BE-RNN. Compared to the DE-RNNR, BE-RNN has significant improvements in performance. Although we are making this comparison, we cannot make the claim that the bi-encoder architecture is always better than the dual encoder architecture because the proposed BE model has different loss function and hyper-parameters than the DE model. Moreover,

Model Id	**Description**	**Recall@1**	**Recall@2**	**Recall@5**
DE	Dual Encoder LSTM (Benchmark)	55.2	72.09	92.43
BE	Bi-Encoder LSTM (Proposed)	**56.88**	**73.24**	**92.86**
DE-RNN	Dual Encoder RNN	37.9	56.00	83.60
DE-RNNR	Dual Encoder RNN Reproduced	30.64	48.37	79.45
BE-RNN	Bi-Encoder RNN	**34.6**	**53.10**	**82.3**

Table 3.5: Comparison of BE model with RNN architecture and DE model with RNN. Results are top-k % accuracy on UDCv2 on the validation set

in order for this claim to be valid all variations of the dual encoder architecture must be compared with the bi-encoder architecture.

For a given NLP task, choice of words embedding to real vector space can affect the performance of a model. Table 3.6 shows the results of using various embedding vectors with the BE model.We first looked at the random embedding and then used the Word2Vec embedding trained on the Ubuntu Dialog Corpus. We also used the pre-trained GloVe embeddding (Mikolov et al., 2013) and ran the model with all four pre-trained GloVe embeddings that are available - (1) Wikipedia - 6B tokens, 400K vocab, uncased, 50d, 100d, 200d, and 300d vectors, (2) Common Crawl - 42B tokens, 1.9M vocab, uncased, 300d vectors, (3) Common Crawl - 840B tokens, 2.2M vocab, cased, 300d vectors, and (4) Twitter - 2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d and 200d vectors. Both pre-trained and trained embeddings on UDC show better results than the random embedding. Between the Word2Vec and GloVe, the Common Crawl 42B embedding of the GloVe shows the best result. The T-SNE plot of Word2Vec embeddings is shown in Figure 3.6. As can be seen in the diagram, similar words (for example - “thank”, “thx” and “ty”) appear embedded close to each other.

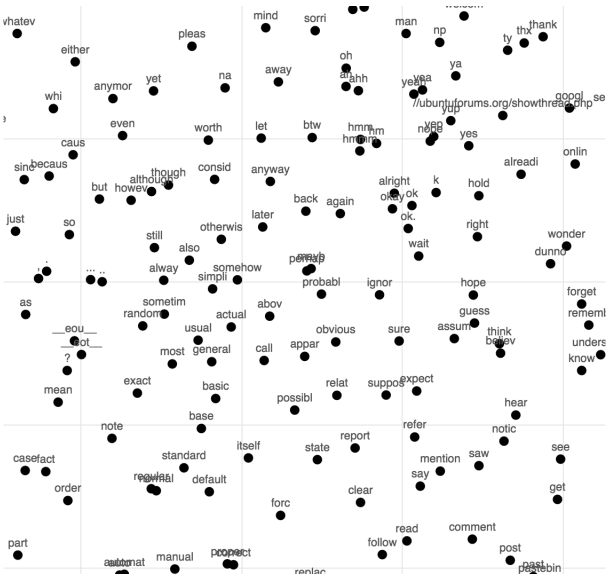


Figure 3.6: T-SNE plot of word embeddings of some frequently occurring words in UDCv2

Embedding	Recall@1	Recall@2	Recall@5
Random	41.7	61.1	87.8
Word2Vec	56.55	73.61	92.7
Twitter 27B 200d	52.50	69.59	91.44
Common Crawl 42B	56.88	73.24	92.86
Common Crawl 840B	56.43	73.25	92.66

Table 3.6: Comparison of performances of the BE model on the with various embedding types. Results are shown on the validation set

In our experiments, we tuned LSTM cell size and the training batch size (Figure 3.7).

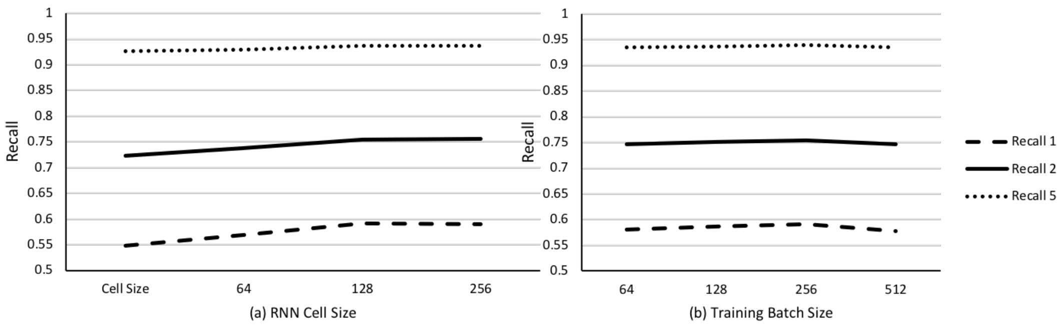


Figure 3.7: Effect of (a) RNN cell size and (b) training batch size on the BE (Bi-Encoder) model

3.3 Error Analysis

Similar to Lowe et al. (2017), we performed qualitative error analysis on the 100 randomly chosen examples from the test dataset where the model made an error for Recall@1 (Table 3.7). The errored examples were evaluated by three persons where each one manually gave a score to each examples for the metrics - Difficulty Rating, Model Response Rating and Error Category.

Difficulty Rating[1-5] measures how difficult human finds the context to match the right response. A rating of 1 on the difficulty scale means that the question is easily answerable by all humans. A 2 indicates moderate difficulty, which should still be answerable by all humans but only if they are paying attention. A 3 means that the question is fairly challenging, and may either require some familiarity with Ubuntu or the human respondent paying very close attention to answer correctly. A 4 is very hard, usually meaning that there are other responses that are nearly as good as the true response; many humans would be unable to answer questions of difficulty 4 correctly. A 5 means that the question is effectively impossible: either the true response is completely unrelated to the context, or it is very short and generic

Model Response Rating[1-3] measures the reasoning of the model's choice. A score of 1 indicates that the model predicted response is completely unreasonable given the context. A 2 means that the response chosen was somewhat reasonable, and that itâĂŹs possible for a human to make a similar mistake. A 3 means that the modelâĂŹs response was more suited to the context than the actual response.

Error Category[1-4] puts model error in a specific category. Error Category of 1 relates to tone and style of the context. If a model makes an error attributed to the misspellings, incorrect grammar, use of emoticons, use of technical jargon or commands etc in the context, then the error category will be 1. Error Category 2 relates to when the context and chosen responses relate to the same topic. Error Category 3 relates to model's inability to account for turn-taking structure. For example if the last turn in the context asks a question and the model chose a answer where it is not answering the question. Error Category 4 means the model picked the response because it sees some common words between the context and the responses.

Difficulty Rating	**% of Errors**
Impossible (5)	19%
Very Difficult (4)	11%
Difficult (3)	22%
Moderate (2)	30%
Easy (1)	18%
Model Response Rating	**% of Errors**
Better than actual (3)	23%
Reasonable (2)	21%
Unreasonable (1)	56%
Error Category	**% of Errors**
Common words (4)	13%
Turn-taking (3)	45%
Same topic (2)	26%
Tone and style (1)	16%

Table 3.7: Qualitative evaluation of the errors from the BE model

The qualitative analysis results (Table 3.7) show that the BE model was not able to predict well the turn-taking structure of the dialogs. A little more than half of the errored examples had the human difficulty level ranging from 3 to 5, and almost half of the model responses in the errored examples were either reasonable or better than the actual response. Appendix B and Appendix C shows examples of test sets used for this analysis

Chapter 4

Conclusions and Future Work

This thesis presented a new LSTM based RNN architecture that can score a set of predefined responses given a context(utterance). Empirically we have shown that on average 92.7%, 73.15% and 56.0% of the time, correct response will be in top 5, top 2 and top 1 correct responses respectively in Ubuntu Dialog Corpus Version 2 exceeding the accuracy of the benchmark model in all three metrics. Ronan and Jason (2008) used a language model with a Rank loss/similarity where he had only positive examples and generated negative examples by corrupting the positive ones. Several other works have shown the Rank loss to be useful in training situations where pairs of correct or incorrect items are to be scored (Yoav Goldberg, 2016). Since UDC dataset matches this scenario, we recommend the future work to explore the BE model with the Rank loss. In a large corpus like UDC where users are seeking help in Ubuntu related problems, it is reasonable to assume that there can be multiple thread of discussion(topics) related to Ubuntu. Identifying the latent topics and grouping the utterances based on topics will allow training an ensemble of BE models. As there is no explicit grouping of the utterance, we plan to identity these hidden topics using Latent Dirichlet Allocation (LDA). Topics distribution of utterances can be used to group them using probabilistic measure of distance. We hypothesize that ensembles of BE models may serve in efficient selection of correct responses. Since the retrieval-based systems have to loop through every single possible responses, if the system needs to go through a very large set, the system may be practically not feasible in production. As shown by (Anjuli et al., 2014) one way to reduce the number of possible responses is through clustering. (Sina et al., 2010) and (Hao et al., 2013) also showed several ways of reducing the large set of possible responses to a smaller set. We intend to apply such ideas in our future work. Moreover, in a multi-turn dialog system capturing longer term context is essential to selecting correct response. Our proposed architecture can be extended to more hierarchical RNN layers, capturing longer context. We plan to investigate this further in conjunction with paragraph vector (Quoc and Tomas, 2008).

Bibliography

Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828. 1

Erik Cambria and Bebo White. 2014. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2):48–57. 1

Rollo Carpenter. 2011. Cleverbot. 39

Kenneth Mark Colby, Sylvia Weber, and Franklin Dennis Hilf. 1971. Artificial paranoia. *Artificial Intelligence*, 2(1):1–25. 36

Koby Crammer and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec):265–292. 6

Christian Darkin, Chris James Hewitt, Joost Korngold, Mark Towse, Peter Reynolds, Simon Tyszko, and Jon Bounds. 2002. IâĂŹm sorry dave, iâĂŹm afraid i canâĂŹt do that. In *After Effects Most Wanted*, pages 129–146. Springer. 16

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941. 10

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407. 43

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211. 33

Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE. 4

John J Godfrey, Edward C Holliman, and Jane McDaniel. 1992. Switchboard: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 517–520. IEEE. 39

Yoav Goldberg and Michael Elhadad. 2008. splitsvm: fast, space-efficient, non-heuristic, polynomial kernel computation for nlp applications. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 237–240. Association for Computational Linguistics. 54

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780. 35

John E Hopcroft. 2013. *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India. 20

Charles Lee Isbell, Michael Kearns, Dave Kormann, Satinder Singh, and Peter Stone. 2000. Cobot in lambdamoo: A social statistics agent. In *AAAI/IAAI*, pages 36–41. 38

Sina Jafarpour, Christopher JC Burges, and Alan Ritter. 2010. Filter, rank, and transfer the knowledge: Learning to chat. *Advances in Ranking*, 10:2329–9290. 38

Rudolf Kadlec, Martin Schmid, and Jan Kleindienst. 2015. Improved deep learning baselines for ubuntu corpus dialogs. *arXiv preprint arXiv:1510.03753*. 38

Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, László Lukács, Marina Ganea, Peter Young, et al. 2016. Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 955–964. ACM. 39

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 10, 47

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2015. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*. 37

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*. 37

Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*. 38

Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909*. 2, 37, 38, 40, 50, 51, 53

Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse*, 8(1):31–65. 37, 59

HP Luhn. 1957. 1 a statistical approach to mechanized encoding. *IBM journal*. 42

Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133. 12

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119. 43, 57

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*. 36

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. 43, 45

Alan Ritter, Colin Cherry, and Bill Dolan. 2010. Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North*

American Chapter of the Association for Computational Linguistics, pages 172–180. Association for Computational Linguistics. 37, 40

Alan Ritter, Colin Cherry, and William B Dolan. 2011. Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pages 583–593. Association for Computational Linguistics. 37, 40

Frank Rosenblatt. 1957. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory. 12

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science. 15

Haşim Sak, Andrew Senior, and Françoise Beaufays. 2014. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*. 35

Gerard Salton. 1971. The smart retrieval system-experiments in automatic document processing. *Englewood Cliffs*. 40

Hinrich Schütze. 1992. Dimensions of meaning. In *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 787–796. IEEE Computer Society Press. 43

Iulian Vlad Serban, Ryan Lowe, Peter Henderson, Laurent Charlin, and Joelle Pineau. 2015. A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*. 37

Lifeng Shang, Zhengdong Lu, and Hang Li. 2015. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364*. 37, 40

Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21. 42

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112. 36

Richard S Wallace. 2008. Alice: Artificial intelligence foundation inc. *Received from: http://www. alicebot. org*. 36

Hao Wang, Zhengdong Lu, Hang Li, and Enhong Chen. 2013. A dataset for research on short-text conversations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 935–945. 38

Joseph Weizenbaum. 1966. ElizaâĂŤa computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45. 36

Jason Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. 2013. The dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 404–413. 39

Zhao Yan, Nan Duan, Junwei Bao, Peng Chen, Ming Zhou, Zhoujun Li, and Jianshe Zhou. 2016. Docchat: An information retrieval approach for chatbot engines using unstructured documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 516–525. 38

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*. 35

Appendix A

Hyper-parameters of the BE Model

Hyper-parameters	**Value**
batch_size	256
shuffle_batch	True
cell_size	256
learning_rate	0.001
n_recurrent_layers	1
optimizer	adam
sort_by_len	False
forget_gate_bias	2.0
is_bidirectional	False
n_recurrent_layers	1
max_seq_len_context	160
max_seq_len_response	80

Table A.1: Hyper-parameters used is the BE Model

Appendix B

An Errored Example - I

Table B.1 shows an error example where a human may assign the Difficulty Rating as 1 (easy) , Model Response Rating as 2 (reasonable) and Model Error Category as 2 (same topic)

Context:	i would go ati over nvidia anyday , but the reason be not straightforward regard whi . __eou__ __eot__ whi ati over nvidia ? __eou__ __eot__
Probabilities:	**Candidate Responses:**
0.98	well , ati be general more open than nvidia to start with . henc whi we have much better open sourc driver for ati than nvidia . on the other hand nvidia 's proprietari driver be suppos better than the ati open sourc one , but this differ have be less and less , and it not true in my person experi (though it probabl be general) . also i 'm a bite of a loyalist , and with the rest of my hardwar be ati/amd , it __eou__
0.15	i 'm use this box just for xbmc and i hear that xbmcbuntu have some veri specif version in play on the ubuntu fork they have.. henc whi i think i 'd go down that rout __eou__
0.13	i think you have a good point for a bugreport __eou__
0.08	gotcha __eou__
0.001	backup be onli good if you can restor them . test , test and test again __eou__ brainpaid - whi doe it matter if you uninstal ? just select differ de at login __eou__
0.003	thank __eou__
0.99	probabl becaus it 's more graphic intens . __eou__
0.01	well , i have and i5 now , and i 's use xfce with gnome-do . suppris ! : d __eou__
0.68	yes ? __eou__
0.42	yes , svn+ssh __eou__

Table B.1: An example from the test set where the BE model makes an error for Recall@1. The first candidate response is the correct response to the given context. The rest of the candidate responses are incorrect responses

Appendix C

An Errored Example - II

Table C.1 shows an error example where human may assign the Difficulty Rating as 5 (difficult) , Model Response Rating as 1 (unreasonable) and Model Error Category as 1 (tone and style)

Context:	hi peopl , could anyon here help me instal a kernel patch for my video driver , i 'm new to this and i do not want to screw thing up . __eou__ __eot__ what patch for what driver ? there ppa 's out for a lot of that stuff __eou__ __eot__
Probabilities:	**Candidate Responses:**
0.022	i 'm run xubuntu but i guess it would be the same . this be to stop a page flip error that fill the hdd with log within 10 mins.. i have the .patch file but dont know what to do with it . ani help would be much appreci __eou__
0.021	ii have n't tri ad deb or deb-src line from newer version , becaus i expect the deb line to break thing . i ca n't account for you be unabl to access newer sourc . my usual method be to instal build-dep from the releas i be run , then apt-get the match sourc , download the newer sourc and patch that __eou__ do you do an apt-get updat after ad the dep-src line (without the deb line) ? __eou__ s/dep/deb/ __eou__
0.0150	no . i ve it and my sound work , also if i set it from alsamix , but with the scrollbar of uniti doe n't work . __eou__ onli " exclud audio " work on bar __eou__
0.592	i chang back to the open sourc driver . i 'm gon na instal proprietari again then tri the display control panel brb __eou__
0.053	other than that , i do n't know , sorri __eou__
0.014	you mean when a termin on your network shut down ? __eou__
0.167	well , tri it ? __eou__
0.594	becaus it 's simplier ? i googl it and bam ! answer be second link __eou__
0.001	i suggest to read a manual for ssh connect , go through googl __eou__ good luck man ! __eou__
0.012	gpart ? __eou__

Table C.1: An example from the test set where the BE model makes an error for Recall@1. The first candidate response is the correct response to the given context. The rest of the candidate responses are incorrect responses

Appendix D

DSTC - 2 Dataset Example

DSTC-2 datset consists of dialogs, each one in its own directory. Every dialog has a âĂŸlog.jsonâĂŹ file, containing a Log Object. Labelled datasets also include a âĂŸlabel.jsonâĂŹ file, a Label Object which annotates the dialog. The userâĂŹs true utterance and action are annotated, as well as the dialog state. The domain of a dataset is described by an ontology object, also distributed in JSON. An example of ’label.json’ file is shown below:

```
{
 "caller-id": "0a45bc863d",
 "turns": [
     {
         "turn-index": 0,
         "goal-labels": {
             "food": "english",
             "area": "centre"
         },
         "transcription": "english food center part
         of town",
         "method-label": "byconstraints",
         "audio-file": "pt344x_0000904_0001214.wav",
         "requested-slots": [],
         "semantics": {
             "json": [
                    {
                        "slots": [
                            [
                                "food",
                                "english"
                            ]
                        ],
                        "act": "inform"
                    },
                    {
                        "slots": [
                            [
                                "area",
                                "centre"
                            ]
                        ],
                        "act": "inform"
                    }
                ],
                "cam": "inform(food=english,area=centre)"
            }
        },
        {
            "turn-index": 1,
            "goal-labels": {
                "food": "english",
                "area": "centre"
            },
            "transcription": "yes",
            "method-label": "byconstraints",
            "audio-file": "pt344x_0001758_0001806.wav",
            "requested-slots": [],
            "semantics": {
                "json": [
            {
                "slots": [],
                "act": "affirm"
            }
        ],
        "cam": "affirm()"
    }
},
{
    "turn-index": 2,
    "goal-labels": {
        "food": "british",
        "area": "centre"
    },
    "transcription": "british food",
    "method-label": "byconstraints",
    "audio-file": "pt344x_0002397_0002501.wav",
    "requested-slots": [],
    "semantics": {
        "json": [
            {
                "slots": [
                    [
                        "food",
                        "british"
                    ]
                ],
                "act": "inform"
            }
        ],
        "cam": "inform(food=british)"
    }
},
{
```

```
            "turn-index": 3,
            "goal-labels": {
                "food": "british",
                "pricerange": "dontcare",
                "area": "centre"
            },
            "transcription": "dont care",
            "method-label": "byconstraints",
            "audio-file": "pt344x_0003096_0003166.wav",
            "requested-slots": [],
            "semantics": {
                "json": [
                    {
                        "slots": [
                            [
                                "this",
                                "dontcare"
                            ]
                        ],
                        "act": "inform"
                    }
                ],
                "cam": "inform(=dontcare)"
            }
        },
        {
            "turn-index": 4,
            "goal-labels": {
                "food": "british",
                "pricerange": "dontcare",
                "area": "centre"
            },
            "transcription": "what is the address
            and phone number",
        "method-label": "byconstraints",
        "audio-file": "pt344x_0003824_0004047.wav",
        "requested-slots": [
            "phone",
            "addr"
        ],
        "semantics": {
            "json": [
                {
                    "slots": [
                        [
                            "slot",
                            "addr"
                        ]
                    ],
                    "act": "request"
                },
                {
                    "slots": [
                        [
                            "slot",
                            "phone"
                        ]
                    ],
                    "act": "request"
                }
            ],
            "cam": "request(addr,phone)"
        }
    },
    {
        "turn-index": 5,
        "goal-labels": {
            "food": "british",
                "pricerange": "dontcare",
                "area": "centre"
            },
            "transcription": "thank you good bye",
            "method-label": "finished",
            "audio-file": "pt344x_0005223_0005324.wav",
            "requested-slots": [],
            "semantics": {
                "json": [
                    {
                        "slots": [],
                        "act": "thankyou"
                    },
                    {
                        "slots": [],
                        "act": "bye"
                    }
                ],
                "cam": "thankyou()|bye()"
            }
        }
    ],
    "task-information": {
        "goal": {
            "text": "Task 02826: You are looking for a
              restaurant in the centre and  it should
              serve english food. If there is no such
              venue how about british type of food.
              You want to know the address and phone
              number.",
            "request-slots": [
                "addr",
                "phone"
            ],
            "constraints": [
                [
                    "food",
                    "british"
                ],
                [
                    "area",
                    "centre"
                ]
            ]
        },
        "feedback": {
            "questionnaire": [
                [
                    "The system understood me well.",
                    "strongly agree"
                ]
            ],
            "comments": null,
            "success": true
        }
    },
    "session-id": "voip-0a45bc863d-20130325_200034"
}
```