Multimedia Tools and Applications
https://doi.org/10.1007/s11042-020-10109-y

Predictive approaches for the UNIX command line: curating and exploiting domain knowledge in semantics deficit data

Thoudam Doren Singh[1] · Abdullah Faiz Ur Rahman Khilji[1] · Divyansha[1] · Apoorva Vikram Singh[2] · Surmila Thokchom[3] · Sivaji Bandyopadhyay[1]

Received: 29 April 2020 / Revised: 1 September 2020 / Accepted: 19 October 2020 /
Published online: 09 November 2020

Abstract

The command line has always been the most efficient method to interact with UNIX flavor based systems while offering a great deal of flexibility and efficiency as preferred by professionals. Such a system is based on manually inputting commands to instruct the computing machine to carry out tasks as desired. This human-computer interface is quite tedious especially for a beginner. And hence, the command line has not been able to garner an overwhelming reception from new users. Therefore, to improve user-friendliness and to mark a step towards a more intuitive command line system, we propose two predictive approaches that can benefit all kinds of users specially the novice ones by integrating into the command line interface. These methods are based on deep learning based predictions. The first approach is based on the sequence to sequence (Seq2seq) model with joint learning by leveraging continuous representations of a self-curated exhaustive knowledge base (KB) comprising an all-inclusive command description to enhance the embedding employed in the model. The other is based on the attention-based transformer architecture where a pretrained model is employed. This allows the model to dynamically evolve over time making it adaptable to different circumstances by learning as the system is being used. To reinforce our idea, we have experimented with our models on three major publicly available Unix command line datasets and have achieved benchmark results using GLoVe and Word2Vec embeddings. Our finding is that the transformer based framework performs better on two different datasets of the three in our experiment in a semantic deficit scenario like UNIX command line prediction. However, Seq2seq based model outperforms bidirectional encoder representations from transformers (BERT) based model on a larger dataset.

Keywords UNIX Command Line Prediction · Knowledge Base · LSTM · GLoVe · Joint Learning · BERT

✉ Thoudam Doren Singh
doren@cse.nits.ac.in

Extended author information available on the last page of the article.

1 Introduction

The work aims at resolving the long-standing plight of unfamiliarity with the command line interface in UNIX based systems. This will not only improve the efficiency of the user but also improve the learning curve. The concerned research work treats the problem of UNIX command line prediction as a sequence prediction problem instead of the traditionally adapted provision of recommendation systems. Due to the absence of semantics between commands, we have employed an enhanced model simultaneously assimilated from a textual corpus and KB to create joint word representation vectors (word embeddings). The learning is extracted from a database of UNIX commands using a probabilistic algorithm. Recurrent neural networks (RNNs) [21] is able to "theoretically" use information from the past in predicting the future. However, plain RNNs suffer from vanishing and exploding gradients problems making them hard to use. Thus, we have used long-short term memory (LSTM) [13] which uses gates to flow gradients back in time, solving the issue of vanishing and exploding gradients. We have also experimented with the attention-based model for predicting the sequence of commands using the BERT [6] next sentence prediction task. To allow the attention-based architecture to understand the UNIX command context, we have pre-trained the model on a large corpus of the UNIX command manual. Employing the transformer-based attention mechanism to the sequence prediction task allowed us to compare both approaches in different experimental settings.

The UNIX command line interface has always been a fundamental segment of distinct computer applications in research, academia, and industrial settings. When a user commences practicing the command line interface, he/she finds it utterly sophisticated to use. Due to the extensive customizations, speed and efficiency of the interface compared to graphical user interface (GUI) but rendering it useless for a beginner who is irrationally scared of inputting commands while forsaking the comfort of using visually appealing workflow. The user does not realize the potential efficiency that lies within the command line. Thus, with the advent of a command line prediction system involving accurate prediction, a GUI prototype of the UNIX shell can be brought in place subsequently realizing the serious necessity of a user-friendly environment for end users.

Our model delves into the user's bash history and learns from it while providing an initial path to the user from the past usage patterns of professionals and scientists. Once sufficient interaction is made with the system, the model can dynamically generate user specific usage patterns. We have also been able to establish a novel method and outperform the 50% threshold of accuracy set by previous works. The former maximum accuracy was accomplished in the work [16] in which [5] was extended to employ consideration of error output with the history of commands and dynamic file name replacement. We have attained 47.9% accuracy using the Seq2seq model and 55.23% using the transformer model on the benchmark dataset of Greenberg [10].

The rest of the paper is structured as follows. Initial sections of 2 and 3 deal with the background of the concerned problem and related works respectively. The Section 4 gives a detailed analysis of the three datasets used for this work. Our novelty is summarized in Section 5. The two main approaches to learn from the command knowledge is described in Section 6. The different embedding techniques employed are explained in Section 7. The outline of the model and the experimental setup is described in Section 8. The result and analysis are discussed in Section 9. Finally, we conclude with discussion and future works in Section 10.

Table 1 Accuracy comparison of related works reported on the Greenberg dataset

S No.	Paper name	Accuracy
1	Predicting UNIX Command Lines: Adjusting to User Patterns, [16]	47.9%
2	Toward An Adaptive Command Line Interface, [4]	45%
3	Predicting UNIX commands using decision tables and decision trees, [8]	42%
4	Predicting Sequences of User Actions, [5]	39.9%
5	The Learning Shell, [14]	36.4%
6	Experiments in UNIX command prediction, [3]	31%
7	User Command Prediction by Graph-Based Induction, [27]	22%

2 Background

The data in the UNIX command line prediction system significantly diversify in terms of characteristics and lack of semantics due to their varied nature of independently carrying out tasks at the fundamental level. The commands for beginners are generally easy to predict as their command set is stereotypically quite small and revolves around fewer commands. As the beginner gains an understanding of the commands, he/she starts using those within his/her cognitive reach quite frequently at the intermediate level. The model needs to train for the dynamic changes and patterns in the user's practice history. As the user proceeds to the expert phase, usage of commands which are relatively rare can be noticed. At this stage, the KB mainly comes into the picture. It harnesses the synonymy it has to offer at the core level of the commands archive and integrates it with embeddings to give the rare commands a chance to exist in the implemented model. It also provides a suitable lexical and semantic context to co-exist in natural text representations.

3 Related work

Investigating ways to customize command line interface has not been a very common practice. In the past, the relevant work [16] discussed three main algorithms to this end. It comprised of C4.5: a well used decision tree algorithm, Most recent command (MRC) that predicts the same command that was just executed and Most frequent command (MFC) which predicts the command that previously occurred most frequently in the user's interactions. The work reported a maximum accuracy of 45% employing a novel use of extracting feature from the frequency of graph-based induction model and use of data dependency. Another attempt by [8] also gives an example of a system in which they had extended [5] while considering error output with the history of commands coupling it with dynamic file name replacement. The maximum accuracy obtained was 47.9%. In general, many of the previous works for which the accuracies are mentioned in Table 1 were based on [5] using the Greenberg[1] dataset. Their work was based on an improvised version of these algorithms as shown by the work [8]. It was based on C4.5, a predictive approach, MRC, MFC and the Longest Matching Prefix. Recently, the work on NL2Bash [19] converts UNIX commands to natural language text using machine learning techniques.

[1] http://saul.cpsc.ucalgary.ca/pmwiki.php/HCIResources/HCIWWWUnixDataSets

Table 2 Dataset statistics

Dataset	Users	Total length	Year
Greenberg	168	197,524	1988
SEA	50	750,000	2001
PU	8	146,970	1997

Our work differs from the above works on mainly two factors. We used a Seq2seq LSTM model and BERT architecture to predict the sequence whose accuracy advances persistently with the amount of data it gathers from the user (since it's a deep learning approach). Secondly, we have also included the KB along with a corpus to ensure more comprehensive embeddings for our model. Using KB and corpus to enhance embeddings is an intuitive idea which has captured research attention recently by the work [1].

4 Dataset

4.1 Data acquisition for knowledge base

We have benchmarked our results on the Greenberg dataset as is the case with the previous works mentioned in Table 1. A detailed dataset statistics is mentioned in Table 2. The data for KB extraction was procured by web scraping from linux.die.net[2] using multiple threads of parallel web computing. A comprehensive database having 38,052 commands was amassed. The data thus attained comprised of particulars regarding command name, an all-inclusive description of the functioning of each command and a salient notion of commands akin to its own. We maneuvered our experiments on the "Unix Data Description dataset by Saul Greenberg" [10] which comprises 168 trace files collected from 168 different users. The data was assembled using 4 bash user groups, comprising 168 overall female and male users with a wide range of technical experience. However, for experimentation enterprise, we utilized the data curated from the group of *52 Computer Scientists*. We substantially preferred this group as tasks performed by this group were least monotonous and more heterogeneous than the rest of the user groups making them more turbulent to predict. The applications involved spanning research investigations, high-end software development, maintaining databases, social communication, word-processing, and fulfilling personal requirements.

We trained and tested our model using a standardized and widely adopted dataset i.e. the one curated by Saul Greenberg [10] as discussed in Section 3. We have also experimented our models on the SEA dataset[3] that focuses on masquerade detection [24] and also on the UNIX dataset proposed by Purdue University (PU) available at University of California, Irvine (UCI) repository [4] [17]. The SEA dataset is prepared by collecting data for months of about 50 users whereas the dataset obtained from PU contains data collected from 8 unique users over a course of up to 2 years. We also considered utilizing the dataset used in [19] but the data here did not suit our purpose as it was not well sanitized. We prioritized Greenberg's data due to its well formatted nature making it better suited to the needs of our

[2]https://linux.die.net/man/

[3]http://www.schonlau.net/intrusion.html

[4]https://archive.ics.uci.edu/ml/datasets/UNIX+User+Data

model. Thus, to verify our claims and reinforce our idea, we have tested our models on all the major publicly available datasets.

4.2 Pre-processing of command sequences

A comprehensive exercise of data pre-processing was implemented to minimize inconsistencies in data. The command lines rendering any grade of error were exterminated from the curated dataset. The instances containing any directories/files after a command were substituted with the keyword “filename” while occurrences specifying any class of parameters were substituted with the keyword “parameter”. The aliases were replaced with the original commands.

4.3 Word cloud visualisation

Word Clouds can be used to visualize the corpus. Here, the word clouds give an inherently knowledgeable overview of a text by depicting the words that occur on the basis of its popularity in the context [11]. As shown in Fig. 1, `ls` and `cd` are the most occurring commands in our pre-processed dataset, following the steps of Section 4.2.

5 Our novelty

Our main concept was to employ a highly promising Seq2seq model to cater to the sequence prediction needs of our project while the other contemplated on the KB. We formulated a way to capture the domain knowledge of UNIX commands focusing on commands which had a rare occurrence in the log of the intermediate user and had frequent appearances in those of specialized scientists. To accomplish this end, we used the KB. We also explored deep-learning techniques to allow our model to learn the domain context from the KB using a deep transformer model. We attained this objective by pretraining BERT on the scrapped manual as discussed in Section 6.2. In short, for learning the knowledge of commands we have used a rule based as well as a deep learning based approach. For the rule based approach, it is noteworthy that it was necessary to normalize the lengths of the command description obtained by the web scraped KB. If it was not carried out, a particular command

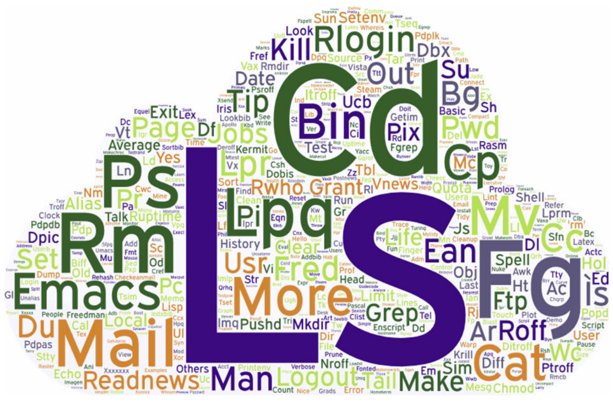

Fig. 1 Word cloud

description would have practically hijacked each and every command description and could have marked its presence as a synonym in the list notwithstanding the available probabilistic inference filter in place.

Further, the commands which were never used in the dataset were omitted from the KB as well due to three main reasons. First, if the commands that were not in the dataset appeared in the KB, it would imply little advantage. Secondly, the time complexity[5] of $\mathcal{O}(n^2)$ wherein the time consumed if all the commands were taken into consideration would be quite large. Therefore, it would be better not to consider each and every command available in the KB. Finally, a good KB should not contain any redundant synonym pairs. The aforementioned limitations were not valid for the deep-learning-based approach using the attention-based transformer model. Thus, for pretraining the BERT model, all the scrapped data was used.

6 Leveraging knowledge base for Seq2seq and transformer based model

To obtain the domain knowledge and understand the contextual information, we use two principal approaches viz. rule based approach and pretraining of transformer based model.

6.1 Rule based approach

To generate the synonyms for the joint learning model the scrapped data were preprocessed by removing all special characters. The preprocessing was carried on for the complete dataset. Stemming was also performed. Also, commonly occurring words having 3-character or less were removed. This data was then used to extract the synonym pairs.

Data, thus obtained, was then iteratively compared with every other data available and five most similar ones were inserted in the list of pairs after they had crossed a particular threshold. The criteria for being similar was taken as the commands which had the most 5 number of exact same words after any pre-processing. The threshold was thus calculated through probabilistic inference considering and analyzing the mean, median and modes of the number of characters available in the data. The algorithm which was designed had to be compatible with parallel computing procedures to enable faster processing on all the processors as well as maintain good hyper-threading performance. To this end, instead of appending all such data into a list and carrying out extraction afterward, it was necessary to implement a one-way procedure to maintain data consistency. The procedure was designed in a fashion in which each thread was given the complete work of initializing, comparing and calculating from the inference as well as printing the synonym pairs of words that would then be fed into the joint learning model. These synonyms were inserted into the joint learning model which in turn facilitated in improving predictions of rare commands by enabling closer placements of its vectors in space.

For constructing an exhaustive KB (as shown in Fig. 2) using a list of vocabulary extracted from the dataset, we designed an algorithm compatible to be run on multiple threads and involving a complete end-to-end approach to cater to the efficiency requirements of the dataset involving 38,052 command elements. The data scrapped from http://linux.die.net contains six different columns namely - ‘Name’, ‘Synopsis’, ‘Description’,

[5]worst case

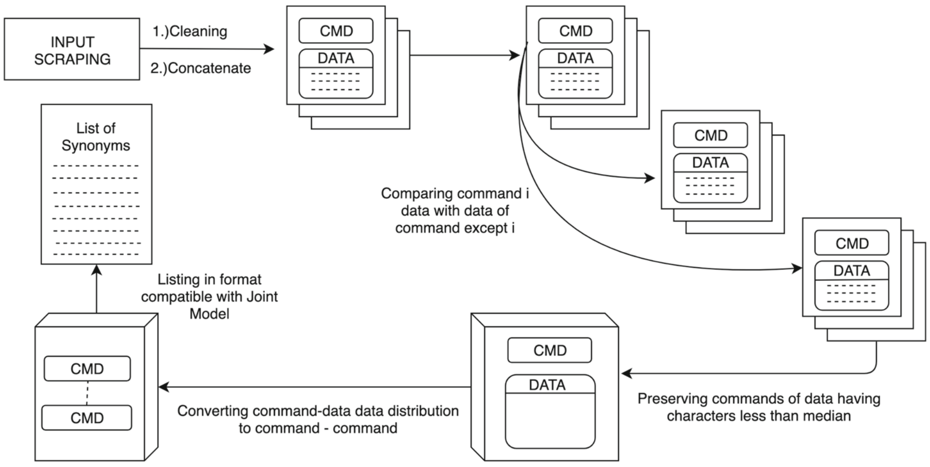
INPUT SCRAPING
1.)Cleaning
2.)Concatenate
CMD
DATA
List of Synonyms
Comparing command i data with data of command except i
Listing in format compatible with Joint Model
Preserving commands of data having characters less than median
Converting command-data data distribution to command - command

Fig. 2 Rule based KB construction

'Options', 'See Also' and 'Referenced by' containing a total of 38,052 rows. The complete data per command was then concatenated, stemmed and all common words containing 3 characters or less were removed. This data was analyzed (referred to as data_concat in this text) further. The average number of characters contained in the data_concat is 3,840.32, with a median of 1,490, maximum of 548,727 and mode of 6,409 characters long (i.e. 128 commands with 6,409 characters). Owing to huge variations in data_concat, normalization of data was carried out. To make the KB more exhaustive and prevent redundancy, we removed all the data outside the vocab (containing 715 commands) on which we would be training later. Still, some commands were found to be practically hijacking the algorithm and labeling themselves as the most similar in most of the cases. Hence, the commands having comparatively large number of words in its manual were selected and the number of words in its manual was shortened.

Finally, for constructing the KB, the data available for each command was compared with every other command and the amount of the same number of words found was stored in an array A. Correspondingly, the top 5 command pairs were printed to a CSV file. All this step of comparing, selecting maximum 5 and printing to a CSV file was carried on from start to end using a single thread to ensure that parallel operation could easily be performed start to end without any conflicts to the file as well as to the compared data which is a typical problem to a parallel computing friendly algorithm.

6.2 Pretraining of transformer based model

For pretraining the BERT model, as will be discussed in Section 8.2 for the predictive approach, the data is made free from all special characters and converted to lower case. Each description of commands is well demarked by spaces to make the model understand the differences between the commands. As compared to different approaches apart from BERT utilizing raw data to learn contextual embeddings, BERT offers a unique advantage to understand bidirectional semantics. In this pretraining step, a deep bidirectional model is used, as shown in Fig. 3.

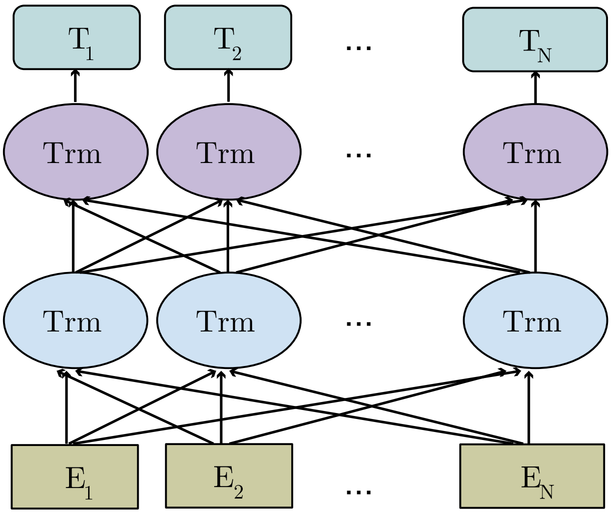

Fig. 3 The bidirectional nature of BERT [6], where the representations are conditioned jointly in all layers on both left and right context

Here, the inputs are passed through the transformer layers (Trm) to get the desirable output having bidirectional context. It is quite evident that such a model is more powerful than a model strictly trained left to right or right to left. To train the model, we mask some tokens for the task also known as Masked Language Modelling (MLM). Thus, to extract the contextual information, we performed MLM also known as the Cloze task [26] on the preprocessed data. The task is to predict randomly masked tokens by masking 15% of the tokens in about 80% of the raw data. To enable the model to handle many downstream tasks effectively, BERT can incorporate multiple sentences using the `[SEP]` token.

As shown in Fig. 4, the representation for the sentence is obtained by adding the corresponding token and the embeddings based on position to the segment. The Segment Embeddings are denoted by E_A and E_B and the Position Embeddings are denoted by E_1, $E_2 \ldots E_n$.

For our use case, we employed BERT with 12 transformer layers pretrained on the BookCorpus dataset. We utilized the pretrained model as the contextual information required to be learned is based on natural language allowing us to get good results without training on for longer duration. The pretraining step was carried out for 100 epochs to obtain an

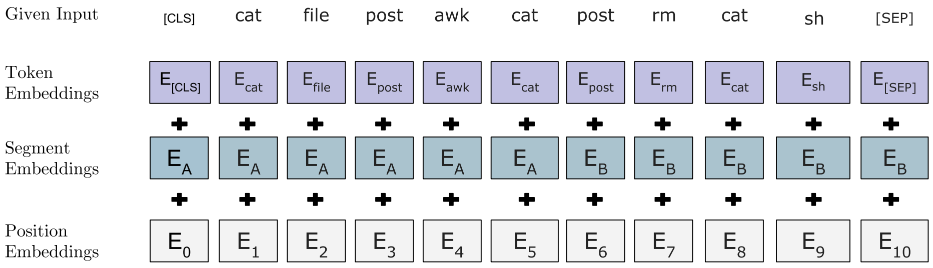

Fig. 4 The given input is the sum total of the token, segment and positional embeddings as employed by BERT [6]

accuracy of 61.18%. The model was then fine-tuned on the specific task of command prediction. Since this model utilizes a self-attention mechanism, it makes easier to accomplish downward natural language processing (NLP) tasks.

7 Command line embedding

In our task, we consider two types of embedding, viz. corpus based word embedding and knowledge based word embedding. These embeddings are used only for Seq2seq based approaches and not for the transformer based approach. For the transformer based approach, we have used the pretraining step as discussed in Section 6.2.

7.1 Corpus based word vector embedding

7.1.1 Word2Vec

Word2Vec [9] attempts to learn a vector representation for every word by using a shallow neural network. It incorporates an input layer, a projection layer, and an output layer for prediction of proximate words that form the basic architecture of the skip-gram model. To augment the log probability of proximate words in the corpus, every word vector is trained for a specific arrangement of words in the sequence $\tau_1, \tau_2 \ldots, \tau_3$ as shown in (1).

$$\frac{1}{\Theta}\sum_{t=1}^{\Theta}\sum_{j\in nb(t)} \log p\left(\tau_j|\tau_t\right) \tag{1}$$

Here, $p(\tau_j|\tau_t)$ denotes the hierarchical softmax of the correlated word v_{τ_t} and v_{τ_j} while nb(t) denotes set of proximate words of word τ_t. The two different learning models introduced as variants of Word2Vec are continuous bag of words (CBOW) and skip-gram.

Continuous Bag of Words (CBOW) Learning This is a prevalent representation technique deployed in information retrieval (IR) and NLP. For this architecture, model is responsible for prediction of current word from a window of neighboring context words. It operates under the assumption that sequence of context words doesn't affect the prediction. CBOW, proposed by Mikolov, transforms similar words to identical vectors and linear operations among these generated vectors to represent the anatomy of the meaning of the words. This model operates by making use of three layers. The first layer is the input layer which corresponds to context. The second layer is the hidden layer which takes each word from the input layer and projects it into the weight matrix which is further projected into the third layer, output layer. The output and the original word is then compared to calculate error gradient which is back propagated to enhance its representation.

Skip-Gram Learning The skip-gram models fetch immense importance in the field of computational linguistics. The architecture of this model is the opposite of CBOW model. Skip-gram aims at prediction of context words in surrounding window by using the current word. The input layer for skip-gram architecture is concerned with the target word while the output layer is concerned with the context. This framework involves calculation of error gradient by comparison between output and every word in context. This error gradient is back propagated through the model to improve representations of words. Owing to its astonishingly straightforward architecture and application of the hierarchical softmax, the training of skip-gram models for very large datasets can be executed at the expense of little computational power.

7.1.2 GLoVe

GloVe [22], an unsupervised learning method for representing words as vectors has been implemented to represent the relationship between two commands. GloVe, short for Global Vectors, trains on word co-occurrence statistics of a textual corpus, with an objective function specially chosen to encode meaningful differences between word vectors. The GloVe predicts surrounding words by maximizing the probability of a context word occurring given a center word by performing a dynamic logistic regression. This architecture utilizes global information while learning various aspects of meaning. The fundamental principle behind working of GloVe can be portrayed as : For two words in a context, the co-occurrence ratio between them is acutely related to meaning.

The activation of GloVe takes off with the generation of a co-occurrence matrix ψ, where each target word,[6] and the context word [7] are represented by rows and columns of ψ respectively.

For ψ_{ij}, $\psi_i = \sum_k \psi_{ik}$ is the frequency of any word in the context of word i. GloVe learns word embedding for every word c_i in vocabulary $\mathcal{V}$ based on whether it is a target word c_i[8] or a context word $\tilde{c}_j$. Finally, the addition of two additional biases b_i for c_i and $\tilde{b}_j$ for $\tilde{c}_j$ is undertaken. The relation between the variables is depicted in (2).

$$J_c = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} f\left(\psi_{ij}\right) \left(c_i^T \tilde{c}_j + b_i + \tilde{b}_j - \log \psi_{ij}\right)^2 \tag{2}$$

7.2 Knowledge based word vector embedding

KB based word vector embeddings and corpus based word vector embeddings are two prevalent stratagems deployed to tackle the word embeddings creation scheme. Although learning word embeddings solely from corpus is a widely practiced approach, it has several limitations. The widely implemented corpus-based learning concepts are based on surface level word co-occurrence matrix disregarding the rich semantic relations between those words. For some instances, the textual corpus might not be adequately large enough to procure reliable word co-occurrence matrix which might pose a problem while learning embeddings for the rare words. On the contrary, KB based approaches operate in quite the divergent manner. KB explicitly defines significance of the words by taking into account the association between words. They include the meaning of those words using semantic relations like synonymy, hypernymy, meronymy, etc. For a KB, every specific word has a finite number of entries, unlike a corpus where several co-occurrences between two words can prevail in dissimilar contexts. Due to this reason, the precise estimation of tenacity of relation between words using KB is not attainable. In a nutshell, the drawback of using corpus-based approach is the omission of rich semantic relations while that of KB is the exclusion of contextual information. These two approaches complement each other. In other words, they deliver on the limitations of each other.

[6]Target word is the word for which we want to learn.

[7]Context word is the one that co-occurs with it in some contextual window

[8]c_i, d denotes the word embedding (vector) of the word $\tilde{c}_j$, and the dimensionality (a user-specified hyperparameter) respectively.

However, since UNIX based commands lack any rank of semantic relations between them, we presumed a definition for the commands to be synonymy of each other. Provided that two commands cross the similarity threshold as specified in 6.1 we consider them as synonyms. Thus, KB will help us in vectorizing the rare commands of the corpus while the corpus will help in vectorization of context-rich commands.

7.3 Comparative analysis of representation techniques

There exist a striking mathematical parallelism between Word2Vec and GloVe irrespective of the diversity in the optimization methods used for them. Word2Vec only differs from GloVe equation with respect to the loss measured between the predicted frequencies and the actual observed frequencies. Word2Vec utilizes cross-entropy loss over the normalized probabilities while GloVe makes use of log mean squared error between actually observed frequencies and the predicted frequencies both being un-normalized. The computational expense of Word2Vec is directly proportional to non-zero elements in word co-occurrence matrix while that of GloVe depends on the size of textual corpus. We have utilised GloVe for the purpose of creating enhanced embeddings by jointly learning through KB and corpora, preferring it over Word2Vec approach since GloVe based embeddings have outperformed Word2Vec based ones as seen in experimental results in Section 9.

7.4 Joint command representation

The objective function J_s harvested using GloVe fails to capture semantic relations between words c_i and $\tilde{c}_j$ as specified by KB. Consequently, it regards all the co-occurrences equitably and encounters fixes when such co-occurrences are rare. To amend this, we derived constraints from the KB that must be satisfied by the learned word embeddings. We defined an objective J_s taking into consideration a KB that not only considers the two-way co-occurrences between c_i and $\tilde{c}_j$ but rather a three-way co-occurrence among c_i, $\tilde{c}_j$ and S [9]. The relation between them is defined in (3).

$$J_s = \sum_{i \in V} \sum_{j \in V} S\left(c_i, \tilde{c}_j\right)\left(c_i - \tilde{c}_j\right)^2 \tag{3}$$

Here, $S(c_i, \tilde{c}_j)$ indicates the strength of the relation S between c_i and $\tilde{c}_j$. If S does not hold between the two words, then $S(c_i, \tilde{c}_j)$ is set to zero. To implement this measure, we have exercised an enhanced model simultaneously assimilating from a text corpus and KB to create joint word representation vectors (word embeddings) [1]. Finally, we used a linearly weighted combination of two objective functions to create a final model of joint embedding incorporating text corpora as well as KB as shown in Fig. 5 and represented by (4).

$$J = J_c + \lambda J_s \tag{4}$$

7.5 Implementation

Considering the commands that occur at least n = 5 times in the corpus, we generated a word co-occurrence matrix ψ. Context window was fixed to m = 50 tokens preceding and succeeding a target word in a sentence and uni-grams were extracted as context words

[9]The semantic relations that exists between the two entities within the KB used in the context.

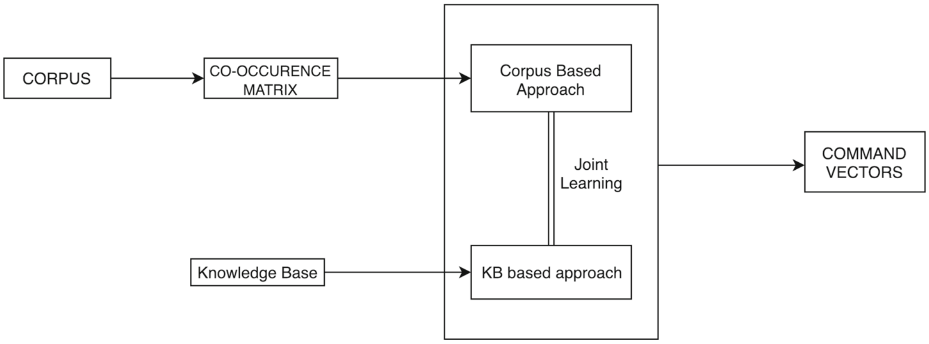

Fig. 5 Joint learning

(commands). The inverse of the distance between the target command and a context command was utilized to weigh the co-occurrences which were measured by the number of tokens appearing in between them. A decreasing function based on weighting was employed using the reciprocal $\frac{1}{d}$ of the distance between two co-occurrences.[10] Stochastic gradient descent (SGD) with learning rate scheduled using AdaGrad [7] as the optimization method was used. The word embeddings were randomly initialized. A consistent distribution range -1 to +1 was used for each dimension distinctly. The initial learning rate in AdaGrad was set to $\alpha = 0.1$ in all of the experiments. Experimentally, z = 100 iterations was established to be ample for the proposed method to converge to a solution. Word co-occurrence matrix ψ stating the co-occurrences between commands in the corpus, dimensionality d = 50 of the word embeddings, relation strength $S(c_i, \tilde{c}_j)$ specifying the extent of synonymy between commands in the KB and the maximum number of iterations z were fed as input. With randomly initialized word vectors $c_i, \tilde{c}_j$, word embeddings of all words $c_i, \tilde{c}_j \in \mathcal{V}$ could be obtained as output after executing the required number of iterations. The iterations are run over all non-zero elements of ψ. Two embeddings c_i (target embedding) and $\tilde{c}_j$ (context embedding) are obtained for each command in $\mathcal{V}$. For post-processing parameters, we derived inspiration from the ensemble method [18]. For this purpose, we applied additive operation on target c_i and context embedding $\tilde{c}_j$ which effectively improves distributive similarity for the embeddings generated, as shown in (5).

$$c = c_i + \tilde{c}_j \tag{5}$$

8 Our approach

8.1 Employing the Seq2seq model as predictive approach

For a small-sized data sample along with a high number of covariates [2], the “small n, large p” problem, prediction becomes challenging. A standard concern for models that strive to make the prediction for a specific user community by utilizing user’s data is the issue of inherent data sparsity. A standard user is liable to generate only a limited quantity of data,

[10]For instance, a context command co-occurring 4 tokens from a target command would impart to a count of $\frac{1}{4}$ to co-occurrence.

despite the massive size of the corpus. To address the subject of inherent data sparsity, various ingenious techniques have been devised.

The approaches undertaken in the past predominantly incorporated probabilistic inference models that have not turned out to be hugely captivating. The major shortcoming faced when this approach is implemented is its inability to appreciate the sequential intelligence of the data. The probabilistic inference method fundamentally makes the prediction by utilizing the number of occurrences of commands in records without acknowledging the sequence in which they have appeared. RNNs architectures offer a fix to this dilemma. Chain-like nature of RNNs reveals that they are intimately connected to list and sequences. This inherent architecture of RNNs makes it an optimal instrument to deal with our sequential data. The Seq2seq model used in our work is shown in Fig. 6.

The Seq2seq is an encoder-decoder architecture where the encoder takes in a fixed size vector and the decoder provides a fixed size vector as output. Both encoders and decoders are based on LSTM cells as discussed in Section 8.1.1 and details of experimental setup are discussed in 8.1.2. A blatant limitation of vanilla neural networks can be attributed to its retention forte, which is highly constrained: it takes as an input, a fixed-sized vector and provides a fixed-sized vector in the form of output. Added to this, these models conduct the mapping using a constant number of computational steps. RNNs provide an edge over this by allowing us to operate on vector sequences : input sequences, output sequences or both (general cases). RNN, a neural architecture that can operate over text sequentially, have shown great promise in dealing with an extensive scope of natural languages processing problems such as parsing, named entity recognition and semantic role labeling. However, a major setback for RNNs is vanishing and exploding gradient problem. The updated values back-propagate through time resulting in multiplication of the gradient value repeatedly. For cases when amplitude of gradient is smaller than one, the repetitive multiplication will propel this value towards zero. Therefore, when we regulate a new value by employing the gradient descent method [25], the model is unable to memorize long-term dependencies.

8.1.1 LSTMs

LSTM [13] provides a solution to the vanishing and exploding gradient problem through the re-parameterization of the RNN. The activation of input gate is multiplied with the input provided to the LSTM cell and preceding values are then multiplied to the forget gate. The network's sole interaction with the LSTM cell occurs via gates. LSTMs perform very well with time series data or with records that are sequential in nature such as user activity or

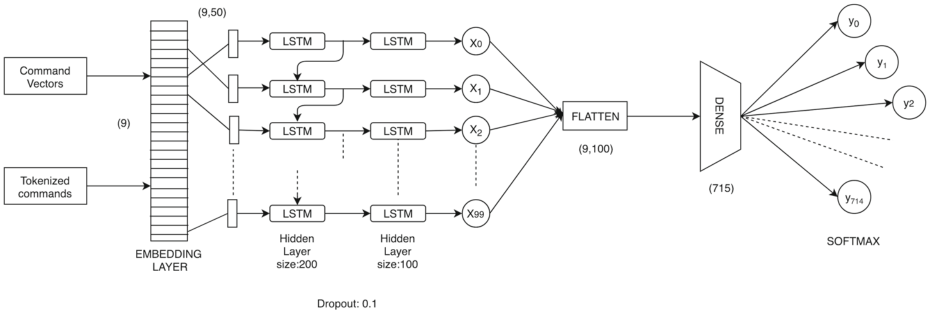

Fig. 6 The Seq2seq Model experimented with Joint Learning, GLoVe and Word2Vec

other events making it competent for principal necessities. LSTM is a modification of RNN possessing an aptitude of handling long-term dependencies. It introduces a new structure in the traditional architecture of RNNs — the memory cell that comprises of four elements: input, forget and output gates and a neuron that links to itself. LSTMs resolve the gradient vanishing problem by retaining the error that can be transmitted by backpropagation through time and layers. In case of LSTM, cell state (C_t) are connected to three gates which are output gate (o_t), input gate (i_t) and forgotten gate (f_t) individually. $\bar{C}_t$ represents the candidate for cell state at timestamp t. b_x denotes the biases for the respective x gates. W_x represents the weights of neurons for the respective x gates. Where, x can be (f) forget gate, (i) input gate, (o) output gate, and (c) cell state vector. h_{t-1} is the output of the previous LSTM block at timestamp $t-1$ and x_t refers to the input at timestamp t. Equations (6), (7), (8), (9), (10) and (11) demonstrate the functions applied.

$$f_{\mathrm{t}} = sigmoid\left(b_{\mathrm{f}} + W_{\mathrm{f}} \cdot \left[h_{\mathrm{t-1}}, x_{\mathrm{t}}\right]\right) \tag{6}$$

$$i_{\mathrm{t}} = sigmoid\left(b_i + W_i \cdot \left[h_{t-1}, x_t\right]\right) \tag{7}$$

$$h_t = o_{\mathrm{t}} * \tanh\left(\mathrm{C_t}\right) \tag{8}$$

$$\bar{C}_t = \tanh\left(\mathrm{b_C} + \mathrm{W_C} \cdot \left[\mathrm{h_{t-1}}, \mathrm{x_t}\right]\right) \tag{9}$$

$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t \tag{10}$$

$$o_t = sigmoid\left(b_0 + W_0 \cdot |h_{t-1}, \mathbf{x}_t|\right) \tag{11}$$

8.1.2 Experimental setup

To make the prediction of the subsequent command using LSTM, the corpus data of commands was tokenized. The tokenized data was then split into training and testing data with the ratio T = 0.9. A vocabulary of size $\mathcal{V}$ = 715 was assembled using tokenized data. Using this vocabulary, the tokenized data and vectors procured using three different mechanisms : Word2Vec, GloVe and Joint Command Learning (as discussed in Section 7.2). We created embedding matrix E to feed in as weights of the embedding layer.

For the model as shown in Fig. 6, we have utilized two LSTM layers, one flatten and one dense layer and finally a softmax layer to make the prediction. The parameters like batch size, number of epochs, optimizer, learning rate and activation function have been tuned to achieve minimum validation loss. To keep a check on over-fitting of data, we have used dropout = 0.1.

In this section, we furnish the empirical testimony that joint command learning based method outperforms baseline Word2Vec as well as baseline GloVe based approaches. We report the experimental results by utilizing three different techniques to create embeddings of the commands : Word2Vec model (skip-gram learning), GloVe and joint command learning i.e. a blend of baseline GloVe and a leveraged KB. The models were evaluated on the dataset retrieved after preprocessing Saul Greenberg's UNIX dataset (Section 4.2). Following pre-processing, a total number 1,67,479 tokens were obtained. For experimental results, we have implemented the three concerned methods to obtain vectors, which are then fed into the LSTM model as discussed in Section 8.1.1. For all the vectors to be created, the dimensionality was set to 50. The first vector creation technique was Word2Vec based approach with skip-gram learning model. For the second method of vector creation, we use GloVe based model to create vectors while setting the context window to be 15 and dimensionality at 50. For the third technique, we use the enhanced joint command representation method (described in detail in Section 7.4) of simultaneously learning from command corpus and KB to obtain vectors by employing GloVe based approach for the corpus segment of the joint learning model. The vectors obtained from the three distinct processes (as mentioned above) were fed into the embedding layer of LSTM engine (elaborated in Section 8.1.1). For batch size in range of 500 to 30000, a curve computing validation loss against number of epochs (in range of 1 to 100) has been plotted for each of the batch size. Minima on the validation loss vs number of epochs curve is assumed to be optimal number of epochs for each batch size i.e. maximum cross validation accuracy was achieved at that particular number of epochs for that batch size. The tuning of final LSTM model involves varying a number of parameters including number of epochs, learning rate, batch size, optimizer and activation function. However, number of epochs and batch size were discovered as most impactful factor. To harvest this observation, the calculation of optimal number of epochs for each batch size has been undertaken. Proper choice of activation function, learning rate and optimizer was made by varying all the parameters at optimum epoch over a range of batch size. The choice of activation function was made from relu, softmax, softplus, softsign, sigmoid and tanh. The sample set of optimizers included of SGD, RMSProp, Adagrad and AdaDelta. For these set of parameters, learning rate was tuned to obtain best results. For evaluation of predicted commands, we have utilised K-fold cross validation over hold-out approach. For a single hold-out set, of which 0.1 dataset is used for testing and 0.9 for training. The test set usually comes out as too small. This introduces a lot of inconsistencies for distinct partitions of dataset that comprise of test and training sets. K-fold cross validation remains a gold standard evaluation methodology that resolves this issue. This technique diminishes the variance by averaging out on K distinct dataset splits. The choice of K is made keeping in mind the bias-variance trade-off associated with its choice. The value of 5 and 10 has been empirically shown to warrant less impact from paramount levels of bias and variance [23].

8.2 Employing the transformer model as predictive approach

For the BERT based model, we utilized the default settings of [6] for the pretraining component. After the unsupervised step Next Sentence Prediction (NSP) task of [6] was used. To attain this objective, the task was taken as a binary Next Commands Prediction (bNCP) task. Under bNCP, half of the actual next commands are random and half are actual commands. This task is related to the learning objectives displayed by [15] and [20]. For choosing the sentence for this training step, 50% of the time the correct next sentence is taken and 50% of the time we take the incorrect sentence, labeled as `IsNext` and `NotNext` respectively. We use this bNCP task to train the model and use it to generate text for command prediction. The formulated bNCP task can be illustrated by the following examples:

Example I:
Input =
```
[CLS] cat file awk [MASK] bash [SEP] cat awk date [MASK] rm
[SEP]
```

Label = `IsNext`

If the next command list is correct, it is labelled as `isNext`. Enabling the model to classify the commands correctly.

Example II:
Input =
```
[CLS] ls cd [MASK] mkdir zsh cpp [SEP] sh [MASK] xrdb cat rm
[SEP]
```

Label = `NotNext`

If the next command list is incorrect, it is labelled as `notNext`. Enabling the model to learn incorrect command predictions.

8.2.1 Experimental setup

The transformer based model for the bNCP task utilized the experimental setup of [6]. Accordingly, the model was trained with a batch size of 256 sequences for 100 epochs for the pretraining step and 24 epochs for the bNCP task. We used Adam optimizer with a learning rate of 0.0001, $\beta1 = 0.9$, $\beta2 = 0.999$, L2 weight decay of 0.01 along with gelu activation [12].

9 Results and analysis

To qualitatively compare and single out an approach for corpus based segment of joint command learning model, we have piloted a meticulous collation exercise as shown in Tables 3 and 4. It can be evidently observed from Tables 3 and 4 that performance of GloVe excels that of Word2Vec by securing a maximum 10-fold cross validation accuracy of 48.50% outmatching the maximum 10-fold cross validation accuracy of 48.33% attained by Word2Vec.

This observational evidence prompts us to utilise GloVe over Word2Vec for the corpus based segment of joint command learning. Notably, the Table 6 provides us with testimony of the fact that joint command learning outperforms Word2Vec as well as GloVe in terms of maximum 10-fold cross validation accuracy by achieving a value of 48.70%. To further verify the results obtained, we calculated 5-fold cross validation accuracies for Word2Vec, GloVe and joint command learning as shown in Tables 3, 4 and 5.

Table 3 Accuracy comparison of Word2Vec (arranged in the order of batch sizes in thousand)

Batch size	Optimal epoch	Accuracy on test set	10-fold cross validation	5-fold cross validation
0.3	3	51.43	48.18	**47.41**
0.5	4	51.09	**48.33**	47.33
1.0	5	**51.67**	48.21	47.25
2.0	6	51.46	48.16	47.40
3.0	9	51.32	48.19	47.24
4.0	10	50.81	48.22	47.37
5.0	13	51.60	48.10	47.38
		Max: 51.67	Max: 48.33	Max: 47.41

The bold entries denote the maximum values

Table 4 Accuracy comparison of GLoVe (arranged in the order of batch sizes in thousand)

Batch size	Optimal epoch	Accuracy on test set	10-fold cross validation	5-fold cross validation
0.3	5	51.88	48.12	**47.80**
0.5	6	51.67	**48.50**	47.55
1.0	10	51.45	48.34	47.63
2.0	17	51.51	48.47	47.45
3.0	13	51.32	48.23	47.52
4.0	24	**51.89**	48.11	47.35
5.0	40	50.88	48.16	47.46
		Max: 51.89	Max: 48.50	Max: 47.80

The bold entries denote the maximum values

The results obtained were in accordance with 10-fold cross validation accuracies. joint command learning tops the maximum 5-fold cross validation accuracies chart by achieving an accuracy of 47.91%. Glove, again outperforms Word2Vec by attaining a maximum 5-fold cross validation accuracy of 47.80% against 47.41%.

In Table 6, we compared the proposed novel joint command learning model with baseline corpus driven approaches for creation of vectors (Word2Vec and GloVe) by incorporating the vectors procured using concerned techniques into the LSTM model (as described in Section 8.1.1). Baseline GloVe based approach yielded a maximum test accuracy of 51.89% excelling the maximum accuracy of 51.67% scored by baseline Word2Vec model on test set. Alternatively, the proposed novel joint command learning technique outperformed the vanilla corpus driven approaches (Word2Vec and GloVe) by achieving a maximum test accuracy of 52.05%. Out of the 10 folds in 10-fold cross validation and 5 folds in 5-fold cross validation, the first two folds and first fold gave a comparatively lesser accuracies (<45%) respectively, since the initial 20% of the training data contained a great deal of randomness. This result validates our hypothesis that inculcation of domain knowledge in corpus driven approaches outperforms generic corpus driven approaches.

Softmax proved to be the best activation function for our experiments. Adam demonstrated better results as compared to other optimizers. Learning rate was tuned for these set of parameters to attain maximum validation accuracy. Best results are achieved at learning rate of 0.1.

Table 5 Accuracy comparison of Joint Learning (arranged in the order of batch sizes in thousand)

Batch size	Optimal epoch	Accuracy on test set	10-fold cross validation	5-fold cross validation
0.3	5	51.92	48.53	47.41
0.5	5	51.59	**48.70**	**47.91**
1.0	35	51.58	48.36	47.69
2.0	56	51.82	48.47	47.74
3.0	9	51.73	48.39	47.85
4.0	64	51.56	48.19	47.73
5.0	12	**52.05**	48.50	47.73
		Max: 52.05	Max: 48.70	Max: 47.91

The bold entries denote the maximum values

Table 6 Accuracy (in Percent) comparison between various Seq2seq approaches employed on the Greenberg dataset

Model	Test accuracy	10-fold	5-fold
Joint Learning	**52.05**	**48.70**	**47.91**
GLoVe	51.89	48.50	47.80
Word2Vec	51.67	48.33	47.41

The bold entries denote the maximum values

As discussed in Section 4, using just one dataset is insufficient to display the advantages of our proposal. Thus, we have trained and tested the above models using the same experimental setup (as discussed in Section 8.1.2) on the other 2 datasets as well namely the PU dataset and the SEA dataset, as these datasets have different characteristics as discussed in Section 4. The results are tabulated in Table 7.

The results obtained from the experimental setup of [6] on the bNCP task is described in Table 7. From this table, it is quite evident that the bNCP task learns the contextual information quite well due to the Cloze task as mentioned in Section 6.2. Thus, it shows good accuracy on small datasets as well. But, as the dataset size increases the LSTM based model proves to be a better option. This can be attributed to its Seq2seq nature. The drastic increase in the dataset size to about 5.1 times to 750,000 (as shown in Table 2) in the case of SEA dataset was highly beneficial for the Seq2seq model. The increase in the dataset also improved the accuracy in the case of the transformer based bNCP task, but the increase is not significant to overcome the joint learning model.

10 Discussion and conclusion

This work has given a novel insight for sequence prediction. In general, where the words greatly lack semantics but can be aided by the inclusion of background knowledge in the form of KB. We introduce a reasonable approach to prepare an exhaustive KB by extracting

Table 7 Accuracy (in Percent) comparison between various Seq2seq and Transformer based approaches used for Greenberg, SEA and PU datasets

Dataset	Model	Test accuracy
Greenberg	**bNCP**	**55.23**
	Joint Learning	52.05
	GLoVe	51.89
	Word2Vec	51.67
SEA	bNCP	60.05
	Joint Learning	**79.15**
	GLoVe	72.87
	Word2Vec	67.23
PU	**bNCP**	**45.19**
	Joint Learning	25.15
	GLoVe	26.60
	Word2Vec	19.12

The bold entries denote the maximum values

knowledge from the data obtained from the source. It is undoubtedly possible to attain better results of the proposed system by including a richer semantic lexicon and better representational techniques. We also compared this knowledge based approach to extract domain knowledge of these commands with a deep learning approach wherein we allow the model to learn the contextual information by itself.

For the knowledge based approach, there is a great need to explore such an algorithm that takes into consideration both the probabilistic and the domain knowledge and cares for the efficiency in marking the evolution for such a system. Thus, this evolution needs a great deal of experimentation as such a change may typically degrade the performance of the model instead of improving it. To achieve further improvement that should be deemed necessary (for the knowledge base) nevertheless, the sophistication involved (thus increasing the complexity of both forms) we decreased the number of parameters and omitted weight consideration to generate multiple iterations of the KB and performed a series of experiments to obtain a general algorithm to produce a self-sufficient KB.

The KB in our case is costly to produce. The process becomes more elaborated when UNIX commands are involved owing to the absence of semantics between them. Therefore, better performances can be achieved in the future by developing a more thorough semantic lexicon that can represent semantic intelligence better.

The attention-based mechanism was also experimented by utilizing the BERT model for the bNCP task. The BERT model has an added advantage that it leverages the KB to such an extent to provide better prediction than the Seq2seq model which learns from the KB. The surprising drawback is that, it was not able to attain the maximum accuracy compared to the Seq2seq model as can be seen particularly from Table 7 for the SEA dataset. Future works might mitigate these challenges.

As shown in Table 7, though the bNCP task utilizing the BERT model shows better accuracy for Greenberg and PU datasets, it surprisingly falls short of the Seq2seq model employing joint learning in the SEA dataset. This may be attributed to the fact that the size of the SEA dataset is quite large compared to the other two datasets as given in Table 2. To leverage larger datasets, joint learning may be employed with the Seq2seq approach in real life production settings. The Seq2seq model with joint learning is quite light on system resources as compared to the bNCP counterpart. A prototype shell can be made in place implementing the incorporated model by deploying a chronic command history of a single user, and thus generating various types of commands and activities a user intends to perform. Moreover, when the user starts off the shell, the prediction of the first command can be shown and a separate model could be trained. This calls for quite a deal of task-specific engineering using the present model based on LSTM for such a problem statement. One of the inevitable reason the other works might not prefer using a neural net for this task despite its accuracy and efficiency might be attributed to the fact that good quality and a higher quantity of data is required which is indeed quite necessary to reach greater accuracy. For making the command line easily approachable to a beginner, it is necessary to implement such a model in an end to end prototype, thus enabling dynamic as well as assistive predictions to the user.

In this work, we proposed and experimented with a unique strategy involving the use of both KB and deep learning approaches to utilize the domain information, Seq2seq and attention-based prediction models while achieving a commendable accuracy yet maintaining a general workflow for the complete work. However, the transformer based model outperformed on two different dataset of the three in the experiment. Thus, the BERT based model is a better choice for a semantic deficit scenario like UNIX command prediction tasks for a smaller dataset as compared to Seq2seq. However, Seq2seq based model outperforms as we increase the dataset size as compared to BERT based model.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

References

1. Alsuhaibani M, Bollegala D, Maehara T, Kawarabayashi K (2018) Jointly learning word embeddings using a corpus and a knowledge base, vol 13
2. Daee P, Peltola T, Soare M, Kaski S (2017) Knowledge elicitation via sequential probabilistic inference for high-dimensional prediction. Mach. Learn. 106(9-10):1599–1620
3. Davison BD, Hirsh H (1997) Experiments in unix command prediction. In: AAAI/IAAI, p 827
4. Davison BD, Hirsh H (1997) Toward an adaptive command line interface. In: HCI (2), pp 505–508
5. Davison BD, Hirsh H (1998) Predicting sequences of user actions. In: Notes of the AAAI/ICML 1998 workshop on predicting the future: AI approaches to time-series analysis, pp 5–12
6. Devlin J, Chang MW, Lee K, Toutanova K (2018) Bert:, Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805
7. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12(Jul):2121–2159
8. Durant KT, Smith MD (2002) Predicting unix commands using decision tables and decision trees. In: WIT transactions on information and communication technologies, p 28
9. Goldberg Y, Levy O (2014) Word2vec explained:, deriving mikolov others.'s negative-sampling word-embedding method. arXiv:1402.3722
10. Greenberg S (1988) Using unix: Collected traces of 168 users
11. Heimerl F, Lohmann S, Lange S, Ertl T (2014) Word cloud explorer: Text analytics based on word clouds. In: System sciences (HICSS), 2014 47th Hawaii international conference on. IEEE, pp 1833–1842
12. Hendrycks D, Gimpel K (2016) Bridging nonlinearities and stochastic regularizers with gaussian error linear units. 1
13. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
14. Jacobs N (2000) The learning shell. In: Adaptive user interfaces, papers from the 2000 AAAI spring symposium, pp 50–53
15. Jernite Y, Bowman SR, Sontag D (2017) Discourse-based objectives for fast unsupervised sentence representation learning. arXiv:1705.00557
16. Korvemaker B, Greiner R (2000) Predicting unix command lines: adjusting to user patterns. In: AAAI/IAAI, pp 230–235
17. Lane T, Brodley CE (1997) An application of machine learning to anomaly detection. In: Proceedings of the 20th national information systems security conference, vol 377, pp 366–380. Baltimore, USA
18. Levy O, Goldberg Y, Dagan I (2015) Improving distributional similarity with lessons learned from word embeddings. Trans Assoc Comput Linguist 3:211–225
19. Lin XV, Wang C, Zettlemoyer L, Ernst MD (2018) Nl2bash:, A corpus and semantic parser for natural language interface to the linux operating system. arXiv:1802.08979
20. Logeswaran L, Lee H (2018) An efficient framework for learning sentence representations. arXiv:1803.02893
21. Mikolov T, Karafiát M., Burget L, Černockỳ J, Khudanpur S (2010) Recurrent neural network based language model. In: Eleventh annual conference of the international speech communication association
22. Pennington J, Socher R, Manning C (2014) Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1532–1543
23. Rodriguez JD, Perez A, Lozano JA (2010) Sensitivity analysis of k-fold cross validation in prediction error estimation. IEEE Trans Pattern Anal Mach Intell 32(3):569–575
24. Schonlau M, DuMouchel W, Ju WH, Karr AF, Theus M, Vardi Y (2001) Computer intrusion: Detecting masquerades. Stat Sci 58–74
25. Shirai K, Sornlertlamvanich V, Marukata S et al (2016) Recurrent neural network with word embedding for complaint classification. In: Proceedings of the third international workshop on worldwide language

service infrastructure and second workshop on open infrastructures and analysis frameworks for human language technologies (WLSI/OIAF4HLT2016), pp 36–43
26. Taylor WL (1953) “Cloze procedure”: A new tool for measuring readability. Journal Q 30(4):415–433
27. Yoshida K (1994) User command prediction by graph-based induction. In: Tools with artificial intelligence, 1994. Proceedings., sixth international conference on. IEEE, pp 732–735

Affiliations

Thoudam Doren Singh[1] · Abdullah Faiz Ur Rahman Khilji[1] · Divyansha[1] · Apoorva Vikram Singh[2] · Surmila Thokchom[3] · Sivaji Bandyopadhyay[1]

Abdullah Faiz Ur Rahman Khilji
abdullah_ug@cse.nits.ac.in

Divyansha
divyansha_ug@cse.nits.ac.in

Apoorva Vikram Singh
apoorva_ug@ee.nits.ac.in

Surmila Thokchom
surmila.thokchom@nitm.ac.in

Sivaji Bandyopadhyay
sivaji.cse.ju@gmail.com

1 Centre for Natural Language Processing (CNLP) and Department of Computer Science and Engineering, National Institute of Technology Silchar, Assam, India

2 Department of Electrical Engineering, National Institute of Technology Silchar, Assam, India

3 Department of Computer Science and Engineering, National Institute of Technology Meghalaya, Meghalaya, India