

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

RECURRENT NEURAL NETWORK FOR TEXT CLASSIFICATION WITH MULTITASK LEARNING

Bona fide record of work done by

BALASURYA S (17Z310)

NITISH C K (17Z333)

RIDHU VARSHINI M (17Z343)

SHAHEER DAWOOD S (17Z346)

VIJAY KUMAR S (17Z359)

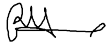
Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of ANNA University

NOVEMBER 2020



.....
[Mr.Marx.R]

Faculty guide

.....
[Dr.Sudha Sadhasivam.G]

Head of the Department

Certified that the candidate was examined in the viva-voce examination held on

.....
(Internal Examiner)

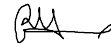
.....
(External Examiner)

CERTIFICATE

Certified that this report titled "**RECURRENT NEURAL NETWORK FOR TEXT CLASSIFICATION WITH MULTITASK LEARNING**", for the Project work-1(15Z720) is a bonafide work of Balasurya.S (17Z310), Nitish.C.K (17Z333), Ridhu varshini.M (17Z343), Shaheer Dawood.S (17Z346) and Vijay Kumar.S (17Z359) who have carried out the work under my supervision for the partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion.

Place: Coimbatore

Date: 18/11/2020



Mr. Marx. R

**Department of Computer Science and
Engineering**

PSG College of Technology

Coimbatore - 641004

COUNTERSIGNED

HEAD

Department of Computer Science and Engineering PSG

College of Technology, Coimbatore - 641004

CONTENTS

CHAPTER	PAGE NO
Acknowledgement.....	(i)
Synopsis.....	(ii)
List of Figures.....	(iii)
List of Tables.....	(iv)
1. INTRODUCTION.....	1
1.1. MOTIVATION.....	1
1.2. PROBLEM STATEMENT	1
1.3. OBJECTIVE	2
1.4. SCOPE OF THE PROJECT.....	2
2. SYSTEM STUDY	3
2.1 INTRODUCTION.....	3
2.2 LITERATURE SURVEY.....	3
2.3 EXISTING ALGORITHMS.....	7
3. SYSTEM ANALYSIS.....	10
3.1. FUNCTIONAL REQUIREMENTS	10
3.2. NON-FUNCTIONAL REQUIREMENTS.....	10
3.3. HARDWARE REQUIREMENTS	11
3.4. SOFTWARE REQUIREMENTS.....	11
3.5. FEASIBILITY ANALYSIS	12
3.5.1. ECONOMICAL FEASIBILITY	12
3.5.2. TECHNICAL FEASIBILITY	12
3.5.3. OPERATIONAL FEASIBILITY	12
4. SYSTEM DESIGN.....	13
4.1. FLOW DIAGRAM	13
4.1.1 MODEL-1:UNIFORM LAYER ARCHITECTURE.....	14
4.1.2 MODEL-2:COUPLED LAYER ARCHITECTURE.....	14
4.1.3 MODEL-3:SHARED LAYER ARCHITECTURE.....	14
4.2. UML ACTIVITY DIAGRAM.....	15
4.3. UML SEQUENCE DIAGRAM.....	16
5. SYSTEM IMPLEMENTATION.....	17
5.1 DATASET INFORMATION	17
5.2 DATA PREPROCESSING	17

5.2.1	TOKENIZING AND PADDING	17
5.3	MODEL-1:UNIFORM LAYER ARCHITECTURE	18
5.4	MODEL-2:COUPLED LAYER ARCHITECTURE	19
5.5	MODEL-3:SHARED LAYER ARCHITECTURE	20
6.	TESTING	21
6.1	MODEL-1:UNIFORM LAYER ARCHITECTURE	21
6.2	MODEL-2:COUPLED LAYER ARCHITECTURE	24
6.3	MODEL-3:SHARED LAYER ARCHITECTURE	26
7.	RESULT	30
8.	CONCLUSION AND FUTURE ENHANCEMENTS.....	31
9.	REFERENCES.....	32
	APPENDIX	33

ACKNOWLEDGEMENT

We express our sincere thanks to our Principal-In-Charge, Dr. K. Prakasan for providing us an opportunity to develop this project.

We pay our deep sense of gratitude to our Head of the Department, Dr. G. Sudha Sadhasivam to encourage us to the highest peak and for providing all the necessary facilities for completing the project.

We express our deepest thanks to our Program Co-Ordinator, Dr. R. Engels and Tutor, Dr. Sathya Priya for extending their friendship towards us and making a pleasure-working environment in all the laboratories.

We also feel to acknowledge our indebtedness and deep sense of gratitude to our guide, Mr. Marx. R, Associate Professor, Department of Computer Science and Engineering whose valuable guidance and kind supervision given to us throughout the project which shaped the present work as it shows.

We express our sincere gratitude to the all the Panel members and faculties-in-charge for assisting and helping us in all possible ways for the completion of this project. We take this opportunity to thank all the teaching and non-teaching staff members of the Department.

SYNOPSIS

Neural network-based methods have obtained great progress on sentiment analysis. In most previous works, the models learn based on single-task supervised objectives and often suffer from insufficient training data. In this project, we propose to use multitask learning framework to jointly learn across multiple related tasks. Three different models with RNN will be integrated into a single system which will be trained jointly using Multi-task learning. Using recurrent neural network, we propose three different mechanisms of sharing information to model text with task-specific and shared layers.

- The first model will use just one shared layer for all the tasks.
- The second model will use different layers for different tasks, but each layer can read information from other layers.
- The third model will assign one specific layer for each task, and also will build a shared layer for all the tasks.

The effectiveness of multi-task learning will be evaluated using four different text classification tasks about movie review with SST-1, SST-2, SUBJ and IMDB movie review and subjectivity datasets. In this project work, we plan to use accuracy as the evaluation metric.

The entire network will be trained jointly on all these tasks. The network will be trained with backpropagation algorithm and the optimisation to be used is gradient-based. The objective is to show that these models can improve the performances of a group of related tasks by exploring common features.

LIST OF FIGURES

FIG NUMBER	FIGURE NAME	PAGE NUMBER
4.1	Flow diagram	13
4.1.1	Uniform layer architecture	14
4.1.2	Coupled layer architecture	14
4.1.3	Shared layer architecture	14
4.2.1	UML Activity diagram -Training	15
4.2.2	UML Activity diagram – Prediction	15
4.3	UML Sequence diagram	16

LIST OF TABLES

TABLE NUMBER	TABLE NAME	PAGE NUMBER
2.1	Existing algorithms	7
3.1	Functional requirements	10
3.2	Non-functional requirements	10
3.3	Hardware requirements	11
3.4	Software requirements	11
6.1	Predictions of Model-1: Uniform layer architecture	21
6.2	Predictions of Model-2: Coupled layer architecture	24
6.3	Predictions of Model-3: Shared layer architecture	26
7.1	Results of Uniform layer architecture	30
7.2	Results of Coupled layer architecture	30
7.3	Results of Shared layer architecture	30

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

The deep neural network-based methods usually need a large-scale corpus due to the large number of parameters; it is hard to train a network that generalizes well with limited data. Specifically, the costs are extremely expensive to build the large-scale resources for typical NLP tasks. Most existing neural network methods are based on supervised training objectives on a single task. These methods often suffer from the limited amounts of training data. To deal with this problem, these models often involve an unsupervised pre-training phase. This unsupervised pretraining is effective to improve the final performance, but it does not directly optimize the desired task.

Multi-task learning utilizes the correlation between related tasks to improve classification by learning tasks in parallel. Motivated by the success of multi-task learning, there are several neural network-based NLP models utilize multitask learning to jointly learn several tasks with the aim of mutual benefit. The basic multi-task architectures of these models are to share some lower layers to determine common features. After the shared layers, the remaining layers are split into the multiple specific tasks.

In this project, we propose three different models of sharing information with recurrent neural network (RNN). All related tasks are integrated into a single system which is trained jointly. Model-I will share a same LSTM layer for different task. In Model-II, there will be a LSTM layer for each task. And Model-III will also have a separate LSTM layer for each task but it is a bidirectional LSTM layer.

1.2 MOTIVATION

The main motive is to use multitask learning framework to jointly learn across multiple related tasks which gives an improved performance in the result where in most previous works, the models are learned based on single-task supervised objectives, which often suffer from insufficient training data.

1.3 PROBLEM STATEMENT

Everyday a very large number of reviews are generated in the internet about a movie. Because of their huge number and size, it is very difficult to handle, analyse and understand such reviews as people just end up with an overload of puzzling feedback that still doesn't answer their questions, unless they devote hours of manual labour to analysing this unstructured data. So, we intend to propose three models which analyse various reviews from four datasets and comes up with a conclusion on whether it's a good movie to watch or not.

1.4 OBJECTIVE

- To propose three multi-task architectures for RNN for classifying movie reviews from four datasets SST-I, SST-II, SUBJ and IMDB.
- To demonstrate results on four text classification tasks.

1.5 SCOPE OF THE PROJECT

The scope of the project is to experiment results on four text classification tasks and show that the joint learning of multiple related tasks together can improve the performance of each task relative to learning them separately.

CHAPTER-2

SYSTEM STUDY

2.1 INTRODUCTION

System study involves studying a procedure or business in order to identify its goals and purposes and create systems and procedures that will achieve them in an efficient way.

2.2 LITERATURE SURVEY

2.2.1 Conneau, Schwenk, Barrault, & Lecun's Very deep convolutional networks for text classification.

In [1], Conneau, Schwenk, Barrault, & Lecun state that the dominant approach for many NLP tasks are recurrent neural networks, in particular LSTMs, and convolutional neural networks, but these architectures are rather shallow in comparison to the deep convolutional networks and present a new architecture (VD-CNN) follows two design principles: which operate at the lowest atomic representation of text, i.e. characters, and use a deep stack of local operations, i.e. convolutions and max-pooling of size 3, to learn a high-level hierarchical representation of a sentence. This architecture has been evaluated on eight freely available large-scale data sets and they were able to show that increasing the depth up to 29 convolutional layers steadily improves performance. Their model generates a 2D tensor of size(f_0 , s) that contains the embeddings of the characters is fixed to 1024, and f_0 can be seen as the "RGB" dimension of the input text. Their models are much deeper than previously published convolutional neural networks and they outperform those approaches on all datasets. To the best of their knowledge, this is the first time that the "benefit of depths" was shown for convolutional neural networks in NLP.

2.2.2 Wei Gao and Fabrizio Sebastiani's from classification to quantification in Tweet Sentiment Analysis - Social Network Analysis and Mining.

In [2], Wei Gao and Fabrizio Sebastiani have dealt with tweet sentiment classification (TSC) using a sub-optimal approach. The reason is that the final goal of most such studies is not estimating the class label (e.g., positive, Negative, or Neutral) of individual tweets, but estimating the relative frequency (a.k.a. "prevalence") of the different classes in the dataset. The latter task is called quantification, and the authors show (by carrying out experiments using two learners, seven quantification-specific algorithms, and eleven TSC datasets) that using quantification-specific algorithms produces substantially better class frequency estimates than a state-of-the-art classification-oriented algorithm routinely used in TSC. They have experimentally shown, on a multiplicity of tweet sentiment classification (TSC) datasets, that more accurate prevalence estimates may be obtained by considering quantification as a task in its own right, i.e., by using quantification-specific learning algorithms which directly attempt to solve the prevalence estimation problem, rather than viewing quantification as a by-product of classification.

2.2.3 Durand, Mehrasa, and Mori's Learning a Deep ConvNet for Multi-Label Classification with Partial Labels.

In [3], Durand, Mehrasa, and Mori first show the algorithm used to predict the classification scores with a GNN in Algorithm 1. The input $x \in \mathbb{R}^C$ of the GNN is the ConvNet output, where C is the number of categories. Most of the existing models to learn with missing labels are

not scalable and do not allow experiments on large-scale dataset like MS COCO and NUS-WIDE. They compare their model with the APG-Graph model that models structured semantic correlations between images on the Pascal VOC 2007 dataset. Unlike their method, the APG-Graph model does not allow to fine-tune the ConvNet. On comparing their model to two baselines: (a) a model trained with the standard BCE where the data are labelled with the partial labels strategy (blue) and (b) a model trained with the standard BCE where the data are labelled with the complete image labels strategy (red). They observe that their model has better performances than the two baselines for most of the metrics. In particular, their final model has significantly better 0-1 exact match performance than the baseline (b), whereas the baseline with partial labels (a) has lower performance than the baseline (b). It is found that the overall precision of the model is worse than the baseline (b), but the overall recall of the model is largely better than the baseline (b).

2.2.4 Daxiang Dong, Hua Wu, Wei He, Dianhai Yu and Haifeng Wang's Multi-Task Learning for Multiple Language Translation.

In [4] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu and Haifeng Wang extend the neural machine translation to a multi-task learning framework which shares source language representation and separates the modelling of different target language translation. Their framework can be applied to situations where either large amounts of parallel data or limited parallel data is available. The model they built is a recurrent neural network-based encoder-decoder model with multiple target tasks, and each task is a specific translation direction. Different tasks share the same translation encoder across different language pairs. The optimization approach used is the mini-batch stochastic gradient descent approach. The difference from the usual stochastic gradient descent is that several minibatches were learnt within a fixed language pair for several mini-batch iterations and then moved onto the next language pair. Their objective is to show that multi-task learning helps to improve translation performance given enough training corpora for all language pairs and for some resource-poor language pairs with a few parallel training data, their translation performance could be improved as well.

2.2.5 Zi-Qiang Wang, Xia Sun, De-Xian Zhang, Xin Li's Optimal SVM-Based Text Classification Algorithm

In [5] Zi-Qiang Wang, Xia Sun, De-Xian Zhang, Xin Li proposed an optimal SVM-Based algorithm for text classification via multiple optimal strategies. The algorithm determines whether a given document belongs to which of the predefined categories. In this SVM-based algorithm, inverse document frequency(I_{df}) is used for measuring the importance of a term in a text document collection. Feature selection is done using the entropy weighting scheme. After the feature selection process, the trained classifier is used for categorizing a new document which adopts SVM for doing the same. Performance is measured using F1 score.

2.2.6 Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang's Representation learning using multi-task deep neural networks for semantic classification and information retrieval.

In [6] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh and Ye-Yi Wang proposed a multitask deep neural network for Query Classification and Web page ranking to optimize the existing vector-based representation. This model maps any arbitrary queries Q or documents D into fixed low dimensional vector representations using DNNs. These vectors can then be used to perform query classification or web search. In contrast to existing representation learning methods which employ either unsupervised or single-task supervised objectives, the proposed DNN model learns these representations using multi-

task objectives. Techniques such as Word Hashing Layer alleviates the problem of representing the words as vectors where vector size depends on the vocabulary size which might be larger in real-time tasks. In order to learn the parameters mini-batch stochastic gradient descent is used.

2.2.7 Orhan Firat, Kyunghyun Cho, and Yoshua Bengio's model for Multi-way, multilingual neural machine translation with a shared attention mechanism.

In [7] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio proposed a model that incorporates attention mechanism to the basic encoder-decoder network. The encoder of the attention-based model encodes a source sentence into a set of context vectors ,whose size varies w.r.t. the length of the source sentence. This context set is constructed by a bidirectional RNN which consists of a forward RNN and reverse RNN. The decoder, which is implemented as an RNN as well, generates one symbol at a time, the translation of the source sentence, based on the context set returned by the encoder. BLEU was used as an evaluation metrics. The proposed model consists of a single neural network that can handle multiple source and target languages simultaneously.

2.2.8 Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean for Efficient estimation of word representations in vector space.

In [8] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean proposed two novel model architectures for computing continuous vector representations of words from very large data sets. A large improvement in accuracy comparing to the previous model is observed at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. A total of four Model Architectures are proposed which are the probabilistic feed forward neural network language model (NNLM), Recurrent neural network-based language model (RNNLM), Continuous Bag-of-Words Model (CBOW) and Continuous Skip-gram Model (CGRAM) and the performance is calculated for both semantic and syntactic tasks. It can be observed that it is possible to train high quality word vectors using very simple model architectures, compared to the popular neural network models (both NNLM and RNNLM).Because of the much lower computational complexity, it is possible to compute very accurate high dimensional word vectors from a much larger data set.

2.2.9 Bo Pang and Lillian Lee for A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts.

In [9] Bo Pang and Lillian Lee proposed a novel machine-learning method that applies text-categorization techniques to just the subjective portions of the document. Document-level polarity classification with a special case of text categorization with sentiment- rather than topic-based categories are to be used. Hence, standard machine learning classification techniques, such as support vector machines (SVMs) or Naive Bayes, can be applied for the classification. The polarity may be improved by removing objective sentences. Therefore, subjectivity detector that determines whether each sentence is subjective or not is implemented: discarding the objective ones creates an extract that should better represent a review's subjective content to a default polarity classifier. As Naive Bayes has somewhat better average ten-fold cross-validation performance on the subjectivity dataset (92% vs. 90%), and so for space reasons, NB subjectivity detection is to be used. Thus, it is examined that the relation between subjectivity detection and polarity classification, showing that ,the subjectivity detection can compress reviews into much shorter extracts that still retain polarity information at a level comparable to that of the full review. In fact, for the Naive Bayes polarity classifier, the subjectivity extracts are shown to be more effective input than

the originating document, which suggests that they are not only shorter, but also “cleaner” representations of the intended polarity.

2.2.10 Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning for Semi-supervised recursive autoencoders for predicting sentiment distributions.

In [10] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng and Christopher D. Manning proposed a novel machine learning framework based on recursive auto encoders for sentence-level prediction of sentiment label distributions. Almost all previous work is based on single, positive/negative categories or scales such as star ratings, etc. Such a one-dimensional scale does not accurately reflect the complexity of human emotions and sentiments. Instead of using a bag-of-words representation, the model exploits hierarchical structure and uses compositional semantics to understand sentiment. The system can be trained both on unlabelled domain data and on supervised sentiment data and does not require any language-specific sentiment lexica, parsers, etc. The approach is based on semi supervised, recursive auto encoders (RAE) which use as input continuous word vectors. The model aims to find vector representations for variable-sized phrases in either unsupervised or semi-supervised training regimes. Thus, without using any hand-engineered resources such as sentiment lexica, parsers or sentiment shifting rules, our model achieves state-of-the-art performance on commonly used sentiment datasets. The evaluation shows that the model can more accurately predict these distributions than other models.

2.2.11 Guimin Jia, Yujun Lu, Weibing Lu, Yihua Shi and Jinfeng Yang’s Verification method for Chinese aviation radiotelephony readbacks based on LSTM-RNN

In [11] the authors propose a novel semantic consistency verification method based on recurrent neural network with long short-term memory structure (LSTM-RNN) for Chinese radiotelephony readbacks. The actual Chinese civil aviation radiotelephony recordings are converted to textual format, and the semantic similarity is studied to verify whether the semantics is the same between the controller instructions and the pilot readbacks. The word-based feature is extracted by one hot vector, and LSTM-RNN is employed to build up a deep network architecture for producing high-level sentence semantic abstraction of the initial input instructions and readbacks pairs. LSTM-RNN adds output gate, forget gate, input gate and memory cell. The memory cell is used to store important information over long-time duration. The output gate, forget gate and input gate of LSTM-RNN are, respectively, used to take itself offline, delete its information and neglect incoming activations. LSTM-RNN not only is affected by the input of previous layer and the previous time state of current layer, as well as the internal state of the LSTM-RNN Cosine similarity is used to quantify the semantic similarity.

2.2.12 Haojin Hu¹ , Mengfan Liao¹ , Chao Zhang¹ , Yanmei Jing² -Text classification based recurrent neural network

The authors combined RNN with LSTM which performs better than traditional RNN and LSTM models. RNN scans the data input mode so that all data parameters of each time step are shared. Each time step will not only receive the input of the current time, but also receive the output of the previous time, thereby successfully using the input and input of the past Information to assist in the judgment of the current moment. For text classification task, RNN represented all sentences to a vector by pre-training , and then, RNN encode all sentences

vector and predict the class of sentence . Although this model can perform text classification well, when faced with long sequence input, RNN suffers from gradient vanishing or exploding, long short-term memory (LSTM) is developed on the basis of Recurrent Neural Network (RNN) to solve these problems.

2.3 Existing Algorithms

STUDIES PRESENTED	ALGORITHM USED	STRENGTH	WEAKNESS	FINDINGS
Very deep convolutional networks for text classification	VD-CNN	Operates directly at the character level and uses only small convolutions and pooling operations	Going even deeper degrades accuracy	a significant improvement on the state-of-the-art configurations can be achieved on text classification tasks by pushing the depth to 29 convolutional layers
Multiple language translation using Multi-task learning	Recurrent neural network-based encoder-decoder	The encoder is shared across different language pairs and each target language has a separate decoder	NIL	Efficient and gets faster and better convergence for both resource-rich and resource-poor language pair under Multi-task learning
semantic classification and information retrieval	Multi-Task Deep neural network	Using Techniques like word hash layer and semantic-representation layer , it shows greater regularization effect. Semantic Representation performs best for small training labels indicates the model's domain adaptation	NIL	Multi-Task learning aims to solve multiple different tasks at the same time, by taking advantage of the similarities between different tasks which shows more regularization than state-of-the-art single-task learning and exhibits more accuracy
Text classification	Support Vector Machine	High generalization ability in case of high dimensional	It underperforms when the data	SVM is preferred in case of classification of

(Document Classification)		data such as text with feature selection which can manage huge feature space and relatively memory efficient	set has more noise i.e. target classes are overlapping	documents where classification is done based on Frequency of words. Term Frequency and Inverse Document Frequency are the techniques that help in making SVM better suited for text and document classification
Multi-way & multilingual Machine translation	Recurrent Neural Network	The modified Autoencoders and Decoders makes the RNN suitable for multiple languages and generalizes better than the single-pair translation model, when the amount of available parallel corpus is small	The number of hyper-parameters grows quadratically with respect to the number of languages	RNN can be used to recognize patterns across time . Gated Recurrent unit and Long Short-Term Memory unit could be used to avoid vanishing gradient problem
Semi-supervised recursive autoencoders for Predicting Sentiment Distributions	Semi-Supervised Recursive Autoencoders	The model exploits hierarchical structure and uses compositional semantics to understand sentiment	NIL	Without using any hand-engineered resources such as sentiment lexica, parsers or sentiment shifting rules, our model achieves state-of-the-art performance on commonly used sentiment datasets
Text Classification	RNN combined with LSTM	RNN suffers from the gradient vanishing and exploding problem while training. This algorithm overcomes it by introducing input	It shows little less accuracy when comparing with Random Multi-model Deep Learning used for	Gradient vanishing problem prevails in most of the RNN architectures which could be overcome by utilizing units such as LSTM and

		gate, forget gate and output gate along with memory cell	classification	GRU
--	--	---	----------------	-----

TABLE 2.1 Existing algorithms

CHAPTER-3

SYSTEM ANALYSIS

System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem-solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose. System Analysis is the process of examining a business situation for the purpose of developing a system solution to a problem or devising improvement to such a situation. It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

3.1 FUNCTIONAL REQUIREMENTS

FUNCTIONS	DESCRIPTION
Model-I	In Model-I, the different tasks share a same LSTM layer and an embedding layer besides their own embedding layers.
Model-II	In Model-II, we assign a LSTM layer for each task, which can use the information for the LSTM layer of the other task.
Model-III	In Model-III, we assign a separate LSTM layer for each task, but introduce a bidirectional LSTM layer to capture the shared information for all the tasks
Training	We choose four different text classification tasks about movie review and train them to show the effectiveness of multi-task learning.
Testing	The data is tested to get the required result about whether to watch the movie or not.

TABLE 3.1 Functional requirements

3.2 NON-FUNCTIONAL REQUIREMENTS

- **Error Analysis** – We analyze most of our bad cases can be generalized into two categories, Complicated sentence structure and Sentence required reasoning.
- **Accuracy** - Our models can improve the performances of a group of related tasks by exploring common features.

3.3 HARDWARE REQUIREMENTS

HARDWARE	DESCRIPTION
Processor	Intel Atom processor or Intel Core i3 processor
Operating system	Operating System : Windows 7 or later
RAM	128 MB or Higher
Disk space	1GB or higher

TABLE 3.2 Hardware requirements

3.4 SOFTWARE REQUIREMENTS

SOFTWARE	DESCRIPTION
Python 3	Language used for programming.
Anaconda 3	A package manager, an environment manager, a Python/R data science distribution, and a collection of over 1,500+ open source packages.
Jupyter Notebook	An open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
Python 3 dependencies	
➤ NumPy	Python library that supports operations on n-D arrays and matrices
➤ Matplotlib	Plotting library for python and its numerical mathematics extension NumPy.
➤ PyTorch	PyTorch is an open source machine learning library used primarily for applications such as computer vision and natural language processing.
➤ TensorFlow	An open source software library for numerical computation using dataflow graphs-structures that describe how data moves through a series of processing nodes.
➤ NLTK	A platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP).
➤ Collections	Python collections module comes with a number of container data types. These data types have different capabilities.

TABLE 3.3 Software requirements

3.5 FEASIBILITY ANALYSIS

A feasibility study is used to determine the practicality of an idea or proposed plan.

3.5.1 Economical Feasibility

All the software dependencies used for this project are open source and not proprietary. So, there was no need of purchasing any software.

3.5.2 Technical Feasibility

TensorFlow is the most used library used in development of Deep Learning models. The community of TensorFlow is extremely vast and supportive, especially because it's an open-sourced platform. On the other hand, NLTK is written in Python and distributed under the GPL open source license.

3.5.3 Operational feasibility

Since the dataset is sentence based and document based, the training can be done with a minimum of 8GB RAM Processor and also it can be trained on Google Colab too.

CHAPTER-4

SYSTEM DESIGN

System Design is the process of defining the architecture, module, interfaces and data for a system to satisfy specified requirements. System design could be seen as the application of systems theory to product development.

4.1 FLOW DIAGRAM

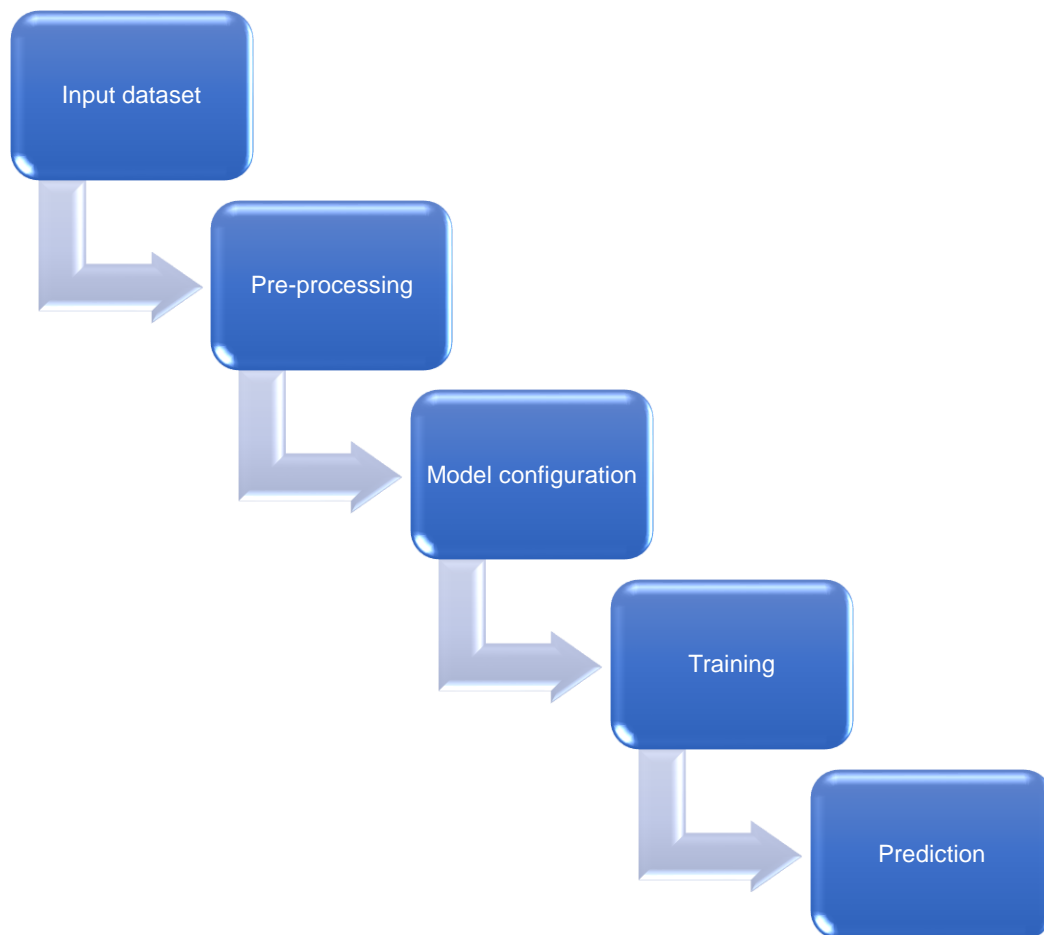


FIG 4.1 Flow diagram

4.1.1 Model-1: Uniform layer Architecture

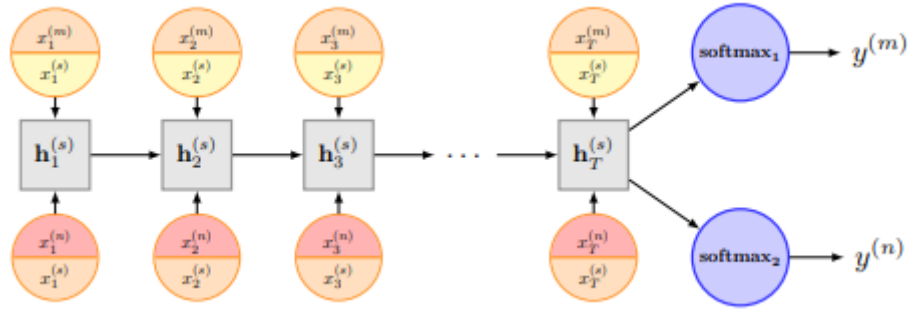


Fig 4.1.1 Uniform layer architecture

4.1.2 Model-2: Coupled layer Architecture

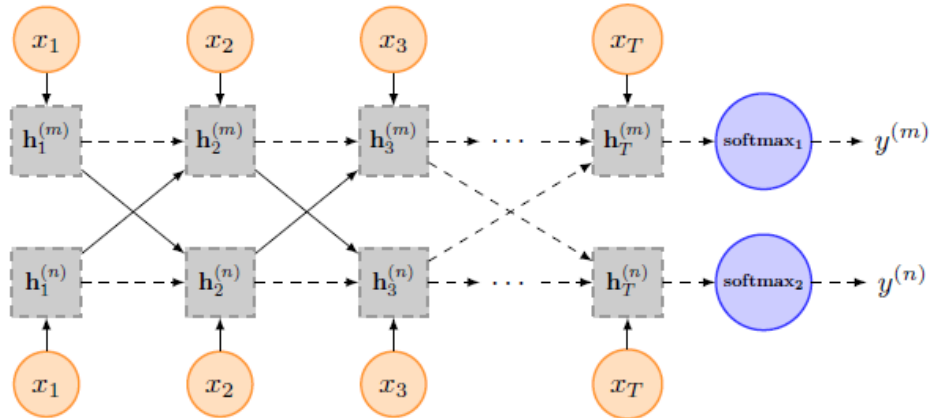


Fig 4.1.2 Coupled layer architecture

4.1.3 Shared layer Architecture

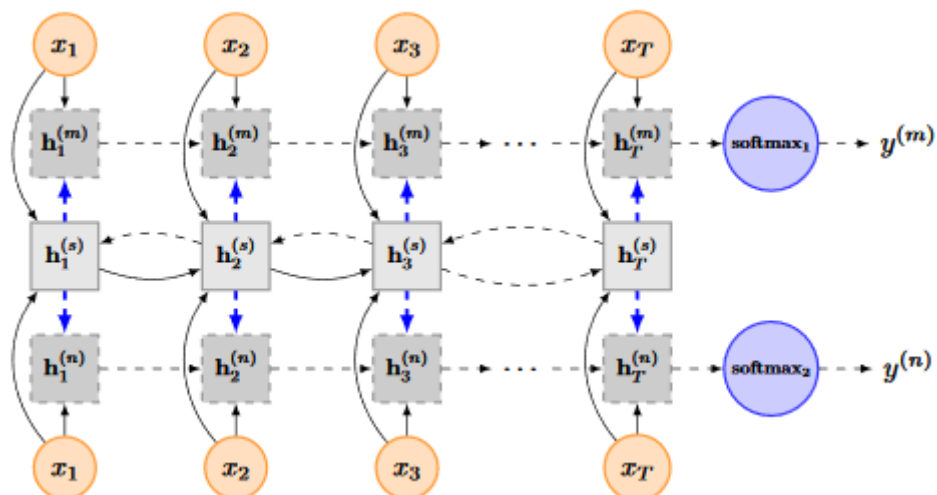


Fig 4.1.3 Shared layer architecture

4.2 UML ACTIVITY DIAGRAM

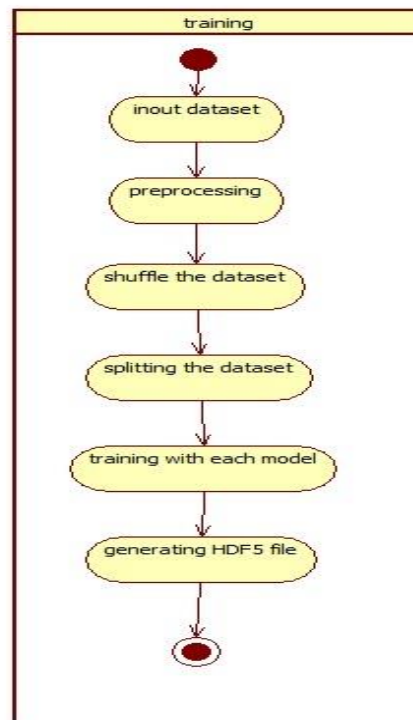


Fig 4.2.1 Activity diagram - training

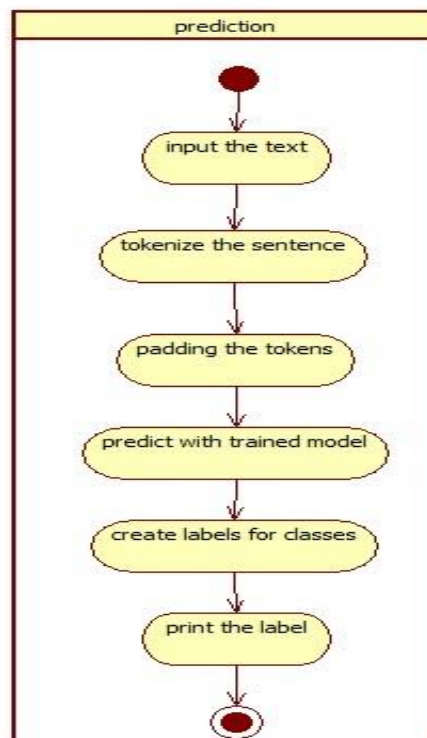


FIG 4.2.2 Activity diagram - Prediction

4.3 UML SEQUENCE DIAGRAM

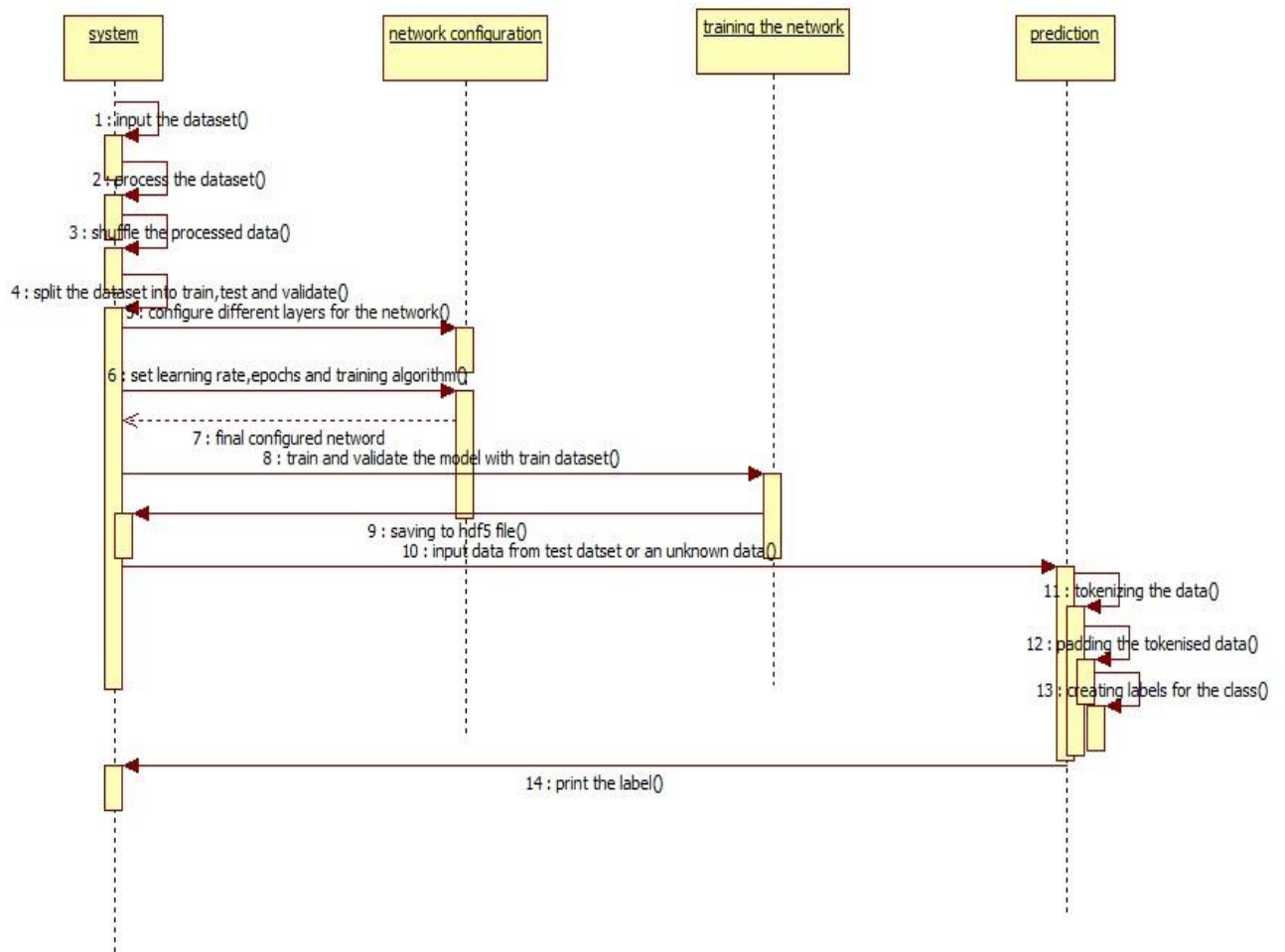


FIG 4.3 Sequence diagram

CHAPTER - 5

SYSTEM IMPLEMENTATION

The Model implementation for text classification using Multi-task learning is explained below. We proposed three Multi-task architectures for RNN for classifying the movie reviews from four datasets SST-1, SST-2, IMDB and SUBJ. Each of the three architectures is different from each other and their implementation is shown below.

5.1 Dataset information

The datasets used for movie review classification are SST-1, SST-2, IMDB and SUBJ.

- SST-1: The Stanford Sentiment Treebank contains 215,154 phrases with fine-grained sentiment labels in the parse trees of 11,855 sentences in movie reviews.
- SST-2: Same as SST-1 but with neutral reviews removed and binary labels.
- SUBJ: Subjectivity dataset where the task is to classify a sentence as being subjective or objective.
- IMDB: Dataset having 50000 movie reviews for natural language processing or Text analytics.

5.2 Data Pre-processing

Data pre-processing includes tokenizing and padding the dataset. Then the dataset is split into train, test and validate.

5.2.1 Tokenizing and Padding

Tokenization chops up a character sequence(sentence) into tokens and throwing away certain characters such as punctuation. We also did padding to ensure that all sequences in the list have the same length.

```
tokenizer = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(concated['review'].values)
sequences = tokenizer.texts_to_sequences(concated['review'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
X = pad_sequences(sequences, maxlen=max_len)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25, random_state=42)
```

5.3 Model-1: Uniform layer architecture

In Uniform layer architecture, each task has a separate Embedding layer. The LSTM layer is shared for all the tasks. The word embeddings are trained using word2Vec.

The hyper parameters used are,

- Embedding size – 64
- Hidden Layer size of LSTM – 64
- Learning rate – 0.1

The summary of Model-1 is,

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 130, 300)	15000000
spatial_dropout1d_1 (Spatial	(None, 130, 300)	0
lstm_1 (LSTM)	(None, 64)	93440
dense_1 (Dense)	(None, 5)	325
Total params: 15,093,765		
Trainable params: 15,093,765		
Non-trainable params: 0		

```
Epoch 1/30
7112/7112 [=====] - 36s 5ms/step - loss: 1.5712 - accuracy: 0.2830 - val_loss: 1.5583 - val_accuracy: 0.2923

Epoch 00001: val_accuracy improved from -inf to 0.29230, saving model to model-1valacc-1.h5
Epoch 2/30
7112/7112 [=====] - 34s 5ms/step - loss: 1.4942 - accuracy: 0.3528 - val_loss: 1.4916 - val_accuracy: 0.3333

Epoch 00002: val_accuracy improved from 0.29230 to 0.33333, saving model to model-1valacc-1.h5
Epoch 3/30
7112/7112 [=====] - 36s 5ms/step - loss: 1.3325 - accuracy: 0.4356 - val_loss: 1.4230 - val_accuracy: 0.3716

Epoch 00003: val_accuracy improved from 0.33333 to 0.37156, saving model to model-1valacc-1.h5
Epoch 4/30
7112/7112 [=====] - 40s 6ms/step - loss: 1.1463 - accuracy: 0.5270 - val_loss: 1.4162 - val_accuracy: 0.3856

Epoch 00004: val_accuracy improved from 0.37156 to 0.38561, saving model to model-1valacc-1.h5
```

5.4 Model-2: Coupled-Layer Architecture

In Model-II, we assign a LSTM layer for each task, which can use the information for the LSTM layer of the other task. This model can be used to jointly learning for every two tasks.

Hyper parameters for the model are as follows,

- Gradient-based optimization is performed using the Adagrad update rule.
- Word embeddings are done using word2vec
 - Vocabulary size - 500000
- Embedding size - 64.
- Hidden layer size of LSTM - 50
- Initial Learning Rate - 0.1

The summary of Model-2 is,

Model: "sequential_4"

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 130, 128)	1024000
spatial_dropout1d_4 (Spatial	(None, 130, 128)	0
lstm_4 (LSTM)	(None, 64)	49408
dense_4 (Dense)	(None, 2)	130
Total params: 1,073,538		
Trainable params: 1,073,538		
Non-trainable params: 0		

Epoch 1/20
6000/6000 [=====] - 23s 4ms/step - loss: 0.6926 - accuracy: 0.5203 - val_loss: 0.6890 - val_accuracy: 0.6110

Epoch 00001: val_accuracy improved from -inf to 0.61100, saving model to model-2valaccbi-115.h5

Epoch 2/20
6000/6000 [=====] - 20s 3ms/step - loss: 0.6842 - accuracy: 0.5933 - val_loss: 0.6758 - val_accuracy: 0.7297

Epoch 00002: val_accuracy improved from 0.61100 to 0.72967, saving model to model-2valaccbi-115.h5

Epoch 3/20
6000/6000 [=====] - 19s 3ms/step - loss: 0.6572 - accuracy: 0.6787 - val_loss: 0.6097 - val_accuracy: 0.8050

Epoch 00003: val_accuracy improved from 0.72967 to 0.80500, saving model to model-2valaccbi-115.h5

Epoch 4/20
6000/6000 [=====] - 20s 3ms/step - loss: 0.5727 - accuracy: 0.7560 - val_loss: 0.4542 - val_accuracy: 0.8370

5.5 Model - 3: Shared layer architecture

Shared layer architecture assigns a separate LSTM layer for each task, but introduces a bidirectional LSTM layer to capture the shared information for all the tasks. To enhance the interaction between task-specific layers and the shared layer, gating mechanism is used to endow the neurons in task-specific layer with the ability to accept or refuse the information passed by the neuron in shared layers.

The hyper parameters used are,

- Embedding size – 64
- Hidden layer size of LSTM – 50
- Learning rate – 0.1
- Regularisation weight – 10^{-5}
- Vocabulary Size – 440000

The summary of Model-3 is,

Model: "sequential_5"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 130, 128)	1024000
spatial_dropout1d_5 (Spatial	(None, 130, 128)	0
dense_5 (Dense)	(None, 130, 2)	258
Total params: 1,024,258		
Trainable params: 1,024,258		
Non-trainable params: 0		

```
Epoch 26/45
23/23 [=====] - 8s 341ms/step - loss: 0.0096 - accuracy: 0.9
Epoch 27/45
23/23 [=====] - 8s 344ms/step - loss: 0.0121 - accuracy: 0.9
Epoch 28/45
23/23 [=====] - 8s 343ms/step - loss: 0.0120 - accuracy: 0.9
Epoch 29/45
23/23 [=====] - 8s 343ms/step - loss: 0.0084 - accuracy: 0.9
Epoch 30/45
23/23 [=====] - 8s 345ms/step - loss: 0.0057 - accuracy: 0.9
Epoch 31/45
23/23 [=====] - 8s 346ms/step - loss: 0.0052 - accuracy: 0.9
Epoch 32/45
23/23 [=====] - 8s 348ms/step - loss: 0.0054 - accuracy: 0.9
Epoch 33/45
23/23 [=====] - 8s 345ms/step - loss: 0.0036 - accuracy: 0.9
Epoch 34/45
23/23 [=====] - 8s 345ms/step - loss: 0.0030 - accuracy: 0.9
Epoch 35/45
23/23 [=====] - 8s 343ms/step - loss: 0.0033 - accuracy: 0.9
Epoch 36/45
23/23 [=====] - 8s 344ms/step - loss: 0.0036 - accuracy: 0.9
Epoch 37/45
23/23 [=====] - 8s 343ms/step - loss: 0.0038 - accuracy: 0.9
Epoch 38/45
23/23 [=====] - 8s 343ms/step - loss: 0.0030 - accuracy: 0.9
```

CHAPTER-6

TESTING

Testing is done separately for all three models. From each task nearly 20 reviews were taken for prediction and we have listed few of those below.

6.1 MODEL-1: UNIFORM LAYER ARCHITECTURE

SST-1			SST-2			IMDB			SUBJ		
REVIEW	EXPEC TED	ACT UAL	REVIEW	EXPEC TED	ACT UAL	REVIEW	EXPEC TED	ACT UAL	REVIEW	EXPEC TED	ACT UAL
this movie is so bad , that it's almost worth seeing because it's so bad .	1	0	the weight of the piece , the unerring professionalism of the chilly production , and the fascination embedded in the lurid topic prove recommendation enough .	1	1	The unthinkable has happened. Having first witnessed it a few years ago, I have had a film that has been my benchmark for awfulness and that film was called "McCinsey's Island"	0	0	they renege , but a legitimate kidney becomes available for transplant .	1	1
much of it is funny , but there is also some startling , surrealistic	3	3	it appears that something has been lost in the translation to the screen .	0	0	I loved this film!! I have been waiting for this film to come out so I could	1	1	when he unexpectedly leaves for a week to go skiing with his wife , question is	1	1

mome nts ...						see it. I saw it opening night and loved it, some people told me that this film wasn't any good and to stay away but I thought it was great.			dragged out on the town by her best friend , Andrea .		
dull , a road trip movie that's surpris ingly short of both advent ure and song .	0	1	the movie is what happens when you blow up small potatoes to 10 times their natural size , and it ai not pretty .	0	0	I once had a convers ation with my parents who told me British cinema goers in the 1940s and 50s would check to see a film's country of origin before going to see it .	0	0	but santa 's got problems (he's even mysterio usly losing weight) and things quickly go south when he finds out that his son , question , has landed on this year 's naughty list .	1	1
it's a gloriou s specta cle like those d.w. questi	4	4	a beguiling splash of pastel colours and prankish comedy	1	0	This movie starts off on the wrong foot and	0	0	rarely , a movie is more than a movie . go .	1	1

on made in the early days of silent film .			from quest .			never really gets it going. The first scene shows a Life Flight helicopter landing and just outside the window you can distinctly see mountains in the background.					
it plays like a big budget , after school special with a generous cast , who at times lift the material from its well question clunkiness .	2	2	you wonder why enough was not just a music video rather than a full-length movie .	0	0	I admit I had no idea what to expect before viewing this highly stylized piece. It could have been the cure for a zombie virus or the common cold for all I knew.	1	0	without getting grand or preachy , the Sprecher's use an unconventional approach to coax us into asking ourselves fundamental question .	0	1

TABLE 6.1 Predictions of Model-1: Uniform layer architecture

6.2 MODEL-2: COUPLED LAYER ARCHITECTURE

SST-1			SST-2			IMDB			SUBJ		
REVIEW	EXPEC TED	ACT UAL	REVIEW	EXPEC TED	ACT UAL	REVIEW	EXPEC TED	ACT UAL	REVIEW	EXPEC TED	ACT UAL
this movie is so bad , that it's almost worth seeing because it's so bad .	1	1	the weight of the piece , the unerring professionalism of the chilly production , and the fascination embedded in the lurid topic prove recommendation enough .	1	0	The unthinkable has happened. Having first witnessed it a few years ago, I have had a film that has been my benchmark for awfulness and that film was called "McCinsey's Island"	0	1	they renege , but a legitimate kidney becomes available for transplant .	1	1
much of it is funny , but there is also some startling , surrealistic moments ...	3	0	it appears that something has been lost in the translation to the screen .	0	0	I loved this film!! I have been waiting for this film to come out so I could see it. I saw it opening night and loved it, some people told me that this	1	0	when he unexpectedly leaves for a week to go skiing with his wife , question is dragged out on the town by her best friend , Andrea .	1	0

						film wasn't any good and to stay away but I thought it was great.					
dull , a road trip movie that's surprisingly short of both adventure and song .	0	1	the movie is what happens when you blow up small potatoes to 10 times their natural size , and it ai not pretty .	0	1	I once had a conversation with my parents who told me British cinema goes in the 1940s and 50s would check to see a film's country of origin before going to see it .	0	0	but santa 's got problems (he's even mysteriously losing weight) and things quickly go south when he finds out that his son , question , has landed on this year 's naughty list .	1	1
it's a glorious spectacle like those d.w. question made in the early days of silent film .	4	0	a beguiling splash of pastel colours and prankish comedy from quest .	1	0	This movie starts off on the wrong foot and never really gets it going. The first scene shows a Life Flight	0	0	rarely , a movie is more than a movie . go .	1	0

						helicopter landing and just outside the window you can distinctly see mountains in the background.					
it plays like a big budget , after school special with a generous cast , who at times lift the material from its well question clunkiness .	2	1	you wonder why enough was not just a music video rather than a full-length movie .	0	1	I admit I had no idea what to expect before viewing this highly stylized piece. It could have been the cure for a zombie virus or the common cold for all I knew.	1	0	without getting grand or preachy , the Sprecher's use an unconventional approach to coax us into asking ourselves fundamental question .	0	1

TABLE 6.2 Predictions of Model-2: Coupled layer architecture

6.3 MODEL-3: SHARED LAYER ARCHITECTURE

SST-1			SST-2			IMDB			SUBJ		
REVIEW	EXPECTED	ACTUAL	REVIEW	EXPECTED	ACTUAL	REVIEW	EXPECTED	ACTUAL	REVIEW	EXPECTED	ACTUAL
this movie is so bad , that it's almost worth seeing	1	1	the weight of the piece , the unerring professionalism of the chilly production	1	0	The unthinkable has happened. Having first	0	1	they renege , but a legitimate kidney becomes available for	1	1

because it's so bad .			, and the fascination embedded in the lurid topic prove recommendation enough .			witnessed it a few years ago, I have had a film that has been my benchmark for awfulness and that film was called "McCinsey's Island"			transplant .		
much of it is funny , but there is also some startling , surrealistic moments ...	3	1	it appears that something has been lost in the translation to the screen .	0	1	I loved this film!! I have been waiting for this film to come out so I could see it. I saw it opening night and loved it, some people told me that this film wasn't any good and to stay away but I thought it was great.	1	1	when he unexpectedly leaves for a week to go skiing with his wife , question is dragged out on the town by her best friend , Andrea .	1	0

dull , a road trip movie that's surprisingly short of both adventure and song .	0	1	the movie is what happens when you blow up small potatoes to 10 times their natural size , and it ai not pretty .	0	1	I once had a conversation with my parents who told me British cinema goers in the 1940s and 50s would check to see a film's country of origin before going to see it .	0	1	but santa 's got problems (he's even mysteriously losing weight) and things quickly go south when he finds out that his son , question , has landed on this year 's naughty list .	1	0
it's a glorious spectacle like those d.w. question made in the early days of silent film .	4	1	a beguiling splash of pastel colours and prankish comedy from quest .	1	1	This movie starts off on the wrong foot and never really gets it going. The first scene shows a Life Flight helicopter landing and just outside the window you can distinctly see mountains in	0	0	rarely , a movie is more than a movie . go .	1	1

						the backgro und.					
it plays like a big budget , after school special with a genero us cast , who at times lift the materi al from its well questi on clunkin ess .	2	1	you wonder why enough was not just a music video rather than a full- length movie .	0	1	I admit I had no idea what to expect before viewing this highly stylized piece. It could have been the cure for a zombie virus or the commo n cold for all I knew.	1	1	without getting grand or preachy , the Sprecher 's use an unconve ntional approach to coax us into asking ourselfe s fundame ntal question .	0	0

TABLE 6.3 Predictions of Model-3: Shared layer architecture

CHAPTER-7

RESULTS

MODEL-1: UNIFORM LAYER ARCHITECTURE

MODEL	SST-1	SST-2	IMDB	SUBJ
RNN-LSTM	41.0	56.6	60.4	59.5
MULTI-TASK LEARNING	41.3	79.6	88.1	91.2

FIG 7.1 Results of Uniform layer architecture

MODEL-2: COUPLED LAYER ARCHITECTURE

MODEL	SST-1	SST-2	IMDB	SUBJ
RNN-LSTM	41.0	56.6	60.4	59.5
SST-1 & SST-2	41.3	82.0	-	-
SST-1 & IMDB	40.3	-	90.9	-
SST-1 & SUBJ	39.2	-	-	92
SST-2 & IMDB	-	83	92.3	-
SST-2 & SUBJ	-	81.3	-	92.1
IMDB & SUBJ	-	-	90.8	91.8

FIG 7.2 Results of Coupled layer architecture

MODEL-3: SHARED LAYER ARCHITECTURE

MODEL	SST-1	SST-2	IMDB	SUBJ
RNN-LSTM	41.0	56.6	60.4	59.5
MULTI-TASK LEARNING	41.1	80.0	90.0	92.1

FIG 7.3 Results of Shared layer architecture

CHAPTER-8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

- ❖ We implemented three multi-task learning architectures for text classification with four types of datasets.
- ❖ The architectures are structurally flexible and can be regarded as a generalized case of many previous works with deliberate designs.
- ❖ The differences among these three architectures is the mechanism of sharing information among the four tasks used.
- ❖ We explore three different scenarios of multi-task learning and our models can improve performances of most tasks with additional related information from others in all scenarios.
- ❖ The sharing mechanisms of the different tasks prove to be efficient and beneficial for text classification.
- ❖ The results of our testing show that these Multi-tasking models can improve the performance of group of related tasks by exploring the common features among the tasks.

8.2 FUTURE ENHANCEMENTS

In future, we would like to investigate the other sharing mechanisms of the different tasks.

CHAPTER 9

REFERENCES

BASE PAPER:

<https://www.ijcai.org/Proceedings/16/Papers/408.pdf>

- [1] Conneau, A., Schwenk, H., Barrault, L., & Lecun, Y. (2016). Very deep convolutional networks for text classification. arXiv preprint arXiv:1606.01781.
- [2] Wei Gao and Fabrizio Sebastiani. 2016. From Classification to Quantification in Tweet Sentiment Analysis. *Social Network Analysis and Mining*, Vol. 6, 19 (2016), 1--22
- [3] Durand T, Mehrasa N, Mori G. Learning a Deep ConvNet for Multi-Label Classification with Partial Labels[J]. 2019
- [4] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation, 2015.
- [5] Zi-Qiang Wang, Xia Sun, De-Xian Zhang and Xin Li. An Optimal SVM-Based Text Classification Algorithm, 2006.
- [6] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval , 2015.
- [7] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. Multi-way, multilingual neural machine translation with a shared attention mechanism, 2016.
- [8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.
- [9] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts, 2004.
- [10] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions, 2011.
- [11] Guimin Jia, Yujun Lu, Weibing Lu, Yihua Shi and Jinfeng Yang's Verification method for Chinese aviation radiotelephony readbacks based on LSTM-RNN.
- [12] Haojin Hu¹, Mengfan Liao¹, Chao Zhang¹, Yanmei Jing² - Text classification based recurrent neural network.

APPENDIX

Uniform-layer-architecture.py:

```
import keras
from keras.layers import Input, LSTM, Dense
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
import numpy as np
import pandas as pd
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.models import Sequential
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.models import load_model
from keras.layers import Bidirectional

data = pd.read_csv("C:/Users/Dell/Downloads/sentiment_dataset-master
(1)/sentiment_dataset-master/SST1.csv", sep=',', delimiter=',', header='infer', names=None)

np.random.seed(1337)

num_of_categories = 45000

shuffled = data.reindex(np.random.permutation(data.index))
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
three= shuffled[shuffled['sentiment'] == 3][:num_of_categories]
two = shuffled[shuffled['sentiment'] == 2][:num_of_categories]
four = shuffled[shuffled['sentiment'] == 4][:num_of_categories]
concat=concated.loc[concated['sentiment'] == 0, 'res'] = 0
```

```

concated.loc[concated['sentiment'] == 1, 'res'] = 1
concated.loc[concated['sentiment'] == 2, 'res'] = 2
concated.loc[concated['sentiment'] == 3, 'res'] = 3
concated.loc[concated['sentiment'] == 4, 'res'] = 4
print(concated['res'][:10])
labels = to_categorical(concated['res'], num_classes=5)
print(labels[:10])
if 'label' in concatед.keys():
    concatед.drop(['label'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(concated['review'].values)
sequences = tokenizer.texts_to_sequences(concated['review'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
X = pad_sequences(sequences, maxlen=max_len)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,
random_state=42)
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train.shape, y_train.shape, X_test.shape, y_test.shape))

d = pd.concat([zero,one,two,three,four], ignore_index=True)
#Shuffle the dataset
concatед = concatед.reindex(np.random.permutation(concatед.index))
concatед['res'] = 0
concatед.loc[concatед['sentiment'] == 0, 'res'] = 0
concatед.loc[concatед['sentiment'] == 1, 'res'] = 1
concatед.loc[concatед['sentiment'] == 2, 'res'] = 2
concatед.loc[concatед['sentiment'] == 3, 'res'] = 3

```

```

concated.loc[concated['sentiment'] == 4, 'res'] = 4
print(concated['res'][:10])
labels = to_categorical(concated['res'], num_classes=5)
print(labels[:10])
if 'label' in concatед.keys():
    concatед.drop(['label'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(concated['review'].values)
sequences = tokenizer.texts_to_sequences(concated['review'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
X = pad_sequences(sequences, maxlen=max_len)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,
random_state=42)
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train.shape, y_train.shape, X_test.shape, y_test.shape))
data1 = pd.read_csv("C:/Users/Dell/Downloads/sentiment_dataset-master
(1)/sentiment_dataset-master/SST2ii.csv", sep=',', delimiter=',', header='infer',
names=None)
np.random.seed(1337)
num_of_categories = 45000
shuffled = data1.reindex(np.random.permutation(data1.index))
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
concatед = pd.concat([one, zero], ignore_index=True)
#Shuffle the dataset
concatед = concatед.reindex(np.random.permutation(concatед.index))
concatед['res'] = 0
concatед.loc[concatед['sentiment'] == 1, 'res'] = 1

```

```

concated.loc[concated['sentiment'] == 0, 'res'] = 0
print(concated['res'][:10])
labels1 = to_categorical(concated['res'], num_classes=2)
print(labels1[:10])
if 'sentiment' in concatd.keys():
    concatd.drop(['sentiment'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer1 = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer1.fit_on_texts(concated['review'].values)
sequences1 = tokenizer1.texts_to_sequences(concated['review'].values)
word_index = tokenizer1.word_index
print('Found %s unique tokens.' % len(word_index))
Y = pad_sequences(sequences1, maxlen=max_len)
X_train1, X_test1, y_train1, y_test1 = train_test_split(Y, labels1, test_size=0.25,
random_state=42)
epochs = 1
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train1.shape, y_train1.shape, X_test1.shape, y_test1.shape))
data2 = pd.read_csv("C:/Users/Dell/Downloads/IMDB Dataset.csv", sep=',', delimiter=',',
header='infer', names=None)
np.random.seed(1337)
num_of_categories = 45000
shuffled = data2.reindex(np.random.permutation(data2.index))
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
concatd = pd.concat([one,zero], ignore_index=True)
#Shuffle the dataset
concatd = concatd.reindex(np.random.permutation(concatd.index))
concatd['res'] = 0
concatd.loc[concatd['sentiment'] == 0, 'res'] = 0

```

```

concated.loc[concated['sentiment'] == 1, 'res'] = 1
print(concated['res'][:10])
labels2 = to_categorical(concated['res'], num_classes=2)
print(labels2[:10])
if 'sentiment' in concatded.keys():
    concatded.drop(['sentiment'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer2 = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer2.fit_on_texts(concatded['review'].values)
sequences2 = tokenizer2.texts_to_sequences(concatded['review'].values)
word_index = tokenizer2.word_index
print('Found %s unique tokens.' % len(word_index))
Z = pad_sequences(sequences2, maxlen=max_len)
X_train2, X_test2, y_train2, y_test2 = train_test_split(Z, labels2, test_size=0.25,
random_state=42)
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape))
data3 = pd.read_csv("C:/Users/Dell/Downloads/sentiment_dataset-master
(1)/sentiment_dataset-master/SUBJ.csv", sep=', ', delimiter=',', header='infer', names=None)
np.random.seed(1337)
num_of_categories = 45000
shuffled = data3.reindex(np.random.permutation(data3.index))
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
concatded = pd.concat([one,zero], ignore_index=True)
#Shuffle the dataset
concatded = concatded.reindex(np.random.permutation(concatded.index))
concatded['res'] = 0
concatded.loc[concatded['sentiment'] == 0, 'res'] = 0
concatded.loc[concatded['sentiment'] == 1, 'res'] = 1

```

```

print(concated['res'][:10])
labels3 = to_categorical(concated['res'], num_classes=2)
print(labels3[:10])
if 'sentiment' in concatd.keys():
    concatd.drop(['sentiment'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer3 = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer3.fit_on_texts(concatd['review'].values)
sequences3 = tokenizer3.texts_to_sequences(concatd['review'].values)
word_index = tokenizer3.word_index
print('Found %s unique tokens.' % len(word_index))
A = pad_sequences(sequences3, maxlen=max_len)
X_train3, X_test3, y_train3, y_test3 = train_test_split(A, labels3, test_size=0.25,
random_state=42)
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train3.shape, y_train3.shape, X_test3.shape, y_test3.shape))
shared_lstm = LSTM(64)
filepath="model-1valaccbi-111.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only
= True,mode='max ')
callbacks_list = [checkpoint]
epochs = 25
model = Sequential()
model.add(Embedding(n_most_common_words, emb_dim, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.3))
model.add(shared_lstm)
model.add(Dense(5, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())

```

```

history = model.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks =callbacks_list)

filepath="model-1valaccbi-222.h5"

checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

model1 = Sequential()

model1.add(Embedding(n_most_common_words, emb_dim, input_length=Y.shape[1]))

model1.add(SpatialDropout1D(0.3))

model1.add(shared_lstm)

model1.add(Dense(2, activation='sigmoid'))

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print(model1.summary())

history1 = model1.fit(X_train1, y_train1, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks =callbacks_list)

filepath="model-1valaccbi-333.h5"

checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

model2 = Sequential()

model2.add(Embedding(n_most_common_words, emb_dim, input_length=Z.shape[1]))

model2.add(SpatialDropout1D(0.3))

model2.add(shared_lstm)

model2.add(Dense(2, activation='sigmoid'))

model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print(model2.summary())

history2 = model2.fit(X_train2, y_train2, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks =callbacks_list)

filepath="model-1valaccbi-444.h5"

checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

model3 = Sequential()

model3.add(Embedding(n_most_common_words, emb_dim, input_length=A.shape[1]))

model3.add(SpatialDropout1D(0.3))

```

```

model3.add(shared_lstm)
model3.add(Dense(2, activation='sigmoid'))
model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model3.summary())

history3 = model3.fit(X_train3, y_train3, epochs=epochs,
batch_size=batch_size, validation_split=0.2, callbacks =callbacks_list)

```

Coupled-layer-architecture.py:

```

import keras

from keras.layers import Input, LSTM, Dense
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.models import Sequential
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.models import load_model
from keras.layers import Bidirectional
import pandas,numpy as np
from sklearn.metrics import accuracy_score,precision_recall_fscore_support
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, Model
from keras.layers.embeddings import Embedding
from keras.layers import Input

```



```

from keras.layers.recurrent import LSTM
from keras.layers import Flatten
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.wrappers import TimeDistributed
from keras.models import load_model
from keras.utils.np_utils import to_categorical
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.optimizers import Adam
from keras.layers.noise import GaussianNoise
from keras.regularizers import l2
from tensorflow.keras.utils import plot_model
from keras.callbacks import TensorBoard

data = pd.read_csv("C:/Users/Dell/Downloads/sentiment_dataset-master
(1)/sentiment_dataset-master/SST1.csv", sep=',', delimiter=',', header='infer', names=None)
np.random.seed(1337)
num_of_categories = 45000
shuffled = data.reindex(np.random.permutation(data.index))
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
three = shuffled[shuffled['sentiment'] == 3][:num_of_categories]
two = shuffled[shuffled['sentiment'] == 2][:num_of_categories]
four = shuffled[shuffled['sentiment'] == 4][:num_of_categories]
concat = concatenated.loc[concatated['sentiment'] == 0, 'res'] = 0
concatated.loc[concatated['sentiment'] == 1, 'res'] = 1
concatated.loc[concatated['sentiment'] == 2, 'res'] = 2
concatated.loc[concatated['sentiment'] == 3, 'res'] = 3
concatated.loc[concatated['sentiment'] == 4, 'res'] = 4
print(concatated['res'][:10])
labels = to_categorical(concatated['res'], num_classes=5)
print(labels[:10])
if 'label' in concatated.keys():
    concatated.drop(['label'], axis=1)

```

```

n_most_common_words = 8000

max_len = 130

tokenizer = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)

tokenizer.fit_on_texts(concated['review'].values)

sequences = tokenizer.texts_to_sequences(concated['review'].values)

word_index = tokenizer.word_index

print('Found %s unique tokens.' % len(word_index))

X = pad_sequences(sequences, maxlen=max_len)

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,
random_state=42)

emb_dim = 128

batch_size = 256

labels[:2]

print((X_train.shape, y_train.shape, X_test.shape, y_test.shape))


d = pd.concat([zero,one,two,three,four], ignore_index=True)

#Shuffle the dataset

concated = concated.reindex(np.random.permutation(concated.index))

concated['res'] = 0

concated.loc[concated['sentiment'] == 0, 'res'] = 0
concated.loc[concated['sentiment'] == 1, 'res'] = 1
concated.loc[concated['sentiment'] == 2, 'res'] = 2
concated.loc[concated['sentiment'] == 3, 'res'] = 3
concated.loc[concated['sentiment'] == 4, 'res'] = 4

print(concated['res'][:10])

labels = to_categorical(concated['res'], num_classes=5)

print(labels[:10])

if 'label' in concated.keys():
    concated.drop(['label'], axis=1)

n_most_common_words = 8000

max_len = 130

tokenizer = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)

```

```

tokenizer.fit_on_texts(concated['review'].values)
sequences = tokenizer.texts_to_sequences(concated['review'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
X = pad_sequences(sequences, maxlen=max_len)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25,
random_state=42)
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train.shape, y_train.shape, X_test.shape, y_test.shape))
data1 = pd.read_csv("C:/Users/Dell/Downloads/sentiment_dataset-master
(1)/sentiment_dataset-master/SST2ii.csv", sep=', ', delimiter=',', header='infer',
names=None)
np.random.seed(1337)
num_of_categories = 45000
shuffled = data1.reindex(np.random.permutation(data1.index))
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
concated = pd.concat([one,zero], ignore_index=True)
#Shuffle the dataset
concated = concatenated.reindex(np.random.permutation(concated.index))
concated['res'] = 0
concated.loc[concated['sentiment'] == 1, 'res'] = 1
concated.loc[concated['sentiment'] == 0, 'res'] = 0
print(concated['res'][:10])
labels1 = to_categorical(concated['res'], num_classes=2)
print(labels1[:10])
if 'sentiment' in concatenated.keys():
    concatenated.drop(['sentiment'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer1 = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)

```

```

tokenizer1.fit_on_texts(concated['review'].values)
sequences1 = tokenizer1.texts_to_sequences(concated['review'].values)
word_index = tokenizer1.word_index
print('Found %s unique tokens.' % len(word_index))
Y = pad_sequences(sequences1, maxlen=max_len)
X_train1, X_test1, y_train1, y_test1 = train_test_split(Y, labels1, test_size=0.25,
random_state=42)
epochs = 1
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train1.shape, y_train1.shape, X_test1.shape, y_test1.shape))
data2 = pd.read_csv("C:/Users/Dell/Downloads/IMDB Dataset.csv", sep=',', delimiter=',',
header='infer', names=None)
np.random.seed(1337)
num_of_categories = 45000
shuffled = data2.reindex(np.random.permutation(data2.index))
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
concated = pd.concat([one, zero], ignore_index=True)
#Shuffle the dataset
concated = concatenated.reindex(np.random.permutation(concated.index))
concated['res'] = 0
concated.loc[concated['sentiment'] == 0, 'res'] = 0
concated.loc[concated['sentiment'] == 1, 'res'] = 1
print(concated['res'][:10])
labels2 = to_categorical(concated['res'], num_classes=2)
print(labels2[:10])
if 'sentiment' in concatenated.keys():
    concatenated.drop(['sentiment'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer2 = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)

```

```

tokenizer2.fit_on_texts(concated['review'].values)
sequences2 = tokenizer2.texts_to_sequences(concated['review'].values)
word_index = tokenizer2.word_index
print('Found %s unique tokens.' % len(word_index))
Z = pad_sequences(sequences2, maxlen=max_len)
X_train2, X_test2, y_train2, y_test2 = train_test_split(Z, labels2, test_size=0.25,
random_state=42)
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape))
data3 = pd.read_csv("C:/Users/Dell/Downloads/sentiment_dataset-master
(1)/sentiment_dataset-master/SUBJ.csv", sep=', ', delimiter=',', header='infer', names=None)
np.random.seed(1337)
num_of_categories = 45000
shuffled = data3.reindex(np.random.permutation(data3.index))
one = shuffled[shuffled['sentiment'] == 1][:num_of_categories]
zero = shuffled[shuffled['sentiment'] == 0][:num_of_categories]
concated = pd.concat([one,zero], ignore_index=True)
#Shuffle the dataset
concated = concatenated.reindex(np.random.permutation(concated.index))
concated['res'] = 0
concated.loc[concated['sentiment'] == 0, 'res'] = 0
concated.loc[concated['sentiment'] == 1, 'res'] = 1
print(concated['res'][:10])
labels3 = to_categorical(concated['res'], num_classes=2)
print(labels3[:10])
if 'sentiment' in concatd.keys():
    concatd.drop(['sentiment'], axis=1)
n_most_common_words = 8000
max_len = 130
tokenizer3 = Tokenizer(num_words=n_most_common_words, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer3.fit_on_texts(concated['review'].values)

```

```

sequences3 = tokenizer3.texts_to_sequences(concatated['review'].values)
word_index = tokenizer3.word_index
print('Found %s unique tokens.' % len(word_index))
A = pad_sequences(sequences3, maxlen=max_len)
X_train3, X_test3, y_train3, y_test3 = train_test_split(A, labels3, test_size=0.25,
random_state=42)
emb_dim = 128
batch_size = 256
labels[:2]
print((X_train3.shape, y_train3.shape, X_test3.shape, y_test3.shape))
units=256
epochs = 25
x = Sequential()
x.add(Embedding(n_most_common_words, emb_dim, input_length=X.shape[1]))
x.add(SpatialDropout1D(0.3))
x.add(LSTM(64))

y = Sequential()
y.add(Embedding(n_most_common_words, emb_dim, input_length=A.shape[1]))
y.add(SpatialDropout1D(0.3))
y.add(LSTM(64))

shared = Dense(units, activation='relu', name='shared')
predictions = Dense(2, activation='sigmoid', name='predictions')

zx = predictions(shared(Dense(units, activation='relu', name='x_limb')(x)))
zy = predictions(shared(Dense(units, activation='relu', name='y_limb')(y)))

model_x = Model(inputs=[x], outputs=[zx])
model_y = Model(inputs=[y], outputs=[zy])
from keras.callbacks import ModelCheckpoint
filepath="model-2valaccbi-111.h5"

```

```

checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

epochs=20

model_x.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

print(model.summary())

history = model_x.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)

filepath="model-2valaccbi-112.h5"

checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

model_y.compile(loss='binary_crossentropy', optimizer=adam,metrics = ['accuracy'] )

tb_cb = TensorBoard(log_dir='./Graph', histogram_freq=0,
                    write_graph=True, write_images=True)

model_history = model_y.fit(x, y=y, batch_size=128, nb_epoch=20,
verbose=1,validation_data = (testx, testy), callbacks =callbacks_list)

units=256

epochs = 25

x = Sequential()

x.add(Embedding(n_most_common_words, emb_dim, input_length=X.shape[1]))

x.add(SpatialDropout1D(0.3))

x.add(LSTM(64))


y = Sequential()

y.add(Embedding(n_most_common_words, emb_dim, input_length=Z.shape[1]))

y.add(SpatialDropout1D(0.3))

y.add(LSTM(64))


shared = Dense(units, activation='relu', name='shared')

predictions = Dense(2, activation='sigmoid', name='predictions')


zx = predictions(shared(Dense(units, activation='relu', name='x_limb')(x)))

zy = predictions(shared(Dense(units, activation='relu', name='y_limb')(y)))

```

```

model_x = Model(inputs=[x], outputs=[zx])
model_y = Model(inputs=[y], outputs=[zy])
from keras.callbacks import ModelCheckpoint
filepath="model-2valaccbi-117.h5"
checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')
callbacks_list = [checkpoint]
epochs=20

```

```

model3_x.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model3_x.summary())
history = model3_x.fit(X_train2, y_train2, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)

```

#run this

```

from keras.callbacks import ModelCheckpoint
filepath="model-2valaccbi-118.h5"
checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')
callbacks_list = [checkpoint]
epochs=22

```

```

model_y.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
history = model_y.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)

```

units=256

epochs = 25

```

x = Sequential()
x.add(Embedding(n_most_common_words, emb_dim, input_length=Y.shape[1]))
x.add(SpatialDropout1D(0.3))
x.add(LSTM(64))

```

```

y = Sequential()
y.add(Embedding(n_most_common_words, emb_dim, input_length=Z.shape[1]))

```



```

y.add(SpatialDropout1D(0.3))
y.add(LSTM(64))

shared = Dense(units, activation='relu', name='shared')
predictions = Dense(2, activation='sigmoid', name='predictions')

zx = predictions(shared(Dense(units, activation='relu', name='x_limb')(x)))
zy = predictions(shared(Dense(units, activation='relu', name='y_limb')(y)))

model_x = Model(inputs=[x], outputs=[zx])
model_y = Model(inputs=[y], outputs=[zy])
from keras.callbacks import ModelCheckpoint
filepath="model-2valaccbi-121.h5"
checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')
callbacks_list = [checkpoint]
epochs=25
model1_x.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model1_x.summary())
history = model1_x.fit(X_train1, y_train1, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)
#run this

from keras.callbacks import ModelCheckpoint
filepath="model-2valaccbi-115.h5"
checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')
callbacks_list = [checkpoint]
epochs=20
model2_y.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model2_y.summary())
history = model2_y.fit(X_train2, y_train2, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)
#run this
#run this

```

```

#run this

#run this

from keras.callbacks import ModelCheckpoint

filepath="model-2valaccbi-115.h5"

checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

epochs=20

model1 = Sequential()

model1.add(Embedding(n_most_common_words, emb_dim, input_length=Y.shape[1]))

model1.add(SpatialDropout1D(0.7))

model1.add(LSTM(64, dropout=0.7, recurrent_dropout=0.7))

model1.add(Dense(2,activation='sigmoid'))

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print(model1.summary())

history = model1.fit(X_train1, y_train1, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)


from keras.callbacks import ModelCheckpoint

filepath="model-2valaccbi-115.h5"

checkpoint = ModelCheckpoint(filepath, monitor ='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

epochs=20

model3 = Sequential()

model3.add(Embedding(n_most_common_words, emb_dim, input_length=A.shape[1]))

model3.add(SpatialDropout1D(0.7))

model3.add(LSTM(64, dropout=0.7, recurrent_dropout=0.7))

model3.add(Dense(2,activation='sigmoid'))

model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print(model3.summary())

history = model3.fit(X_train3, y_train3, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)

```

```

from keras.callbacks import ModelCheckpoint

filepath="model-2valaccbi-115.h5"

checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

epochs=20

model2 = Sequential()

model2.add(Embedding(n_most_common_words, emb_dim, input_length=Z.shape[1]))

model2.add(SpatialDropout1D(0.7))

model2.add(LSTM(64, dropout=0.7, recurrent_dropout=0.7))

model2.add(Dense(2,activation='sigmoid'))

model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print(model1.summary())

history = model2.fit(X_train2, y_train2, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)

```

```

from keras.callbacks import ModelCheckpoint

filepath="model-2valaccbi-115.h5"

checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only
= True,mode='max ')

callbacks_list = [checkpoint]

epochs=20

model3 = Sequential()

model3.add(Embedding(n_most_common_words, emb_dim, input_length=A.shape[1]))

model3.add(SpatialDropout1D(0.7))

model3.add(LSTM(64, dropout=0.7, recurrent_dropout=0.7))

model3.add(Dense(2,activation='sigmoid'))

model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print(model3.summary())

history = model3.fit(X_train3, y_train3, epochs=epochs,
batch_size=batch_size,validation_split=0.2,callbacks=callbacks_list)

```

Shared – layer – architecture.py:

```
import keras
from keras.layers import Input, Concatenate, average, LSTM, Dense, Flatten
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D, Bidirectional, Flatten
from keras.models import Sequential
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
```

```
data = pd.read_csv("SST1 (1).csv", sep=',', delimiter=',', header='infer', names=None)
```

```
data1 = pd.read_csv("SST2ii.csv", sep=',', delimiter=',', header='infer', names=None)
```

```
num_of_categories1 = 45000
shuffled1 = data.reindex(np.random.permutation(data.index))
zero1 = shuffled1[shuffled1['sentiment'] == 0][:num_of_categories1 ]
one1 = shuffled1[shuffled1['sentiment'] == 1][:num_of_categories1 ]
three1 = shuffled1[shuffled1['sentiment'] == 3][:num_of_categories1 ]
two1 = shuffled1[shuffled1['sentiment'] == 2][:num_of_categories1 ]
four1 = shuffled1[shuffled1['sentiment'] == 4][:num_of_categories1 ]
```

```
num_of_categories2 = 45000
shuffled2 = data1.reindex(np.random.permutation(data1.index))
zero2 = shuffled2[shuffled2['sentiment'] == 0][:num_of_categories2]
one2 = shuffled2[shuffled2['sentiment'] == 1][:num_of_categories2]
```

```
concat1 = pd.concat([zero1, one1, two1, three1, four1], ignore_index=True)
#Shuffle the dataset
concat1 = concat1.reindex(np.random.permutation(concat1.index))
concat1['res1'] = 0
```

```
concat2 = pd.concat([one2, zero2], ignore_index=True)
#Shuffle the dataset
concat2 = concat2.reindex(np.random.permutation(concat2.index))
concat2['res2'] = 0
```

```

concated1.loc[concated1['sentiment'] == 0, 'res1'] = 0
concated1.loc[concated1['sentiment'] == 1, 'res1'] = 1
concated1.loc[concated1['sentiment'] == 2, 'res1'] = 2
concated1.loc[concated1['sentiment'] == 3, 'res1'] = 3
concated1.loc[concated1['sentiment'] == 4, 'res1'] = 4

concated2.loc[concated2['sentiment'] == 0, 'res2'] = 0
concated2.loc[concated2['sentiment'] == 1, 'res2'] = 1

print(concated1['res1'][:10])
labels1 = to_categorical(concated1['res1'], num_classes=5)
print(labels1[:10])
if 'label' in concat1.keys():
    concat1.drop(['label'], axis=1)

print(concated2['res2'][:10])
labels2 = to_categorical(concated2['res2'], num_classes=5)
print(labels2[:10])
if 1 in concat2.keys():
    concat2.drop([1], axis=1)

n_most_common_words1 = 8000
max_len1 = 130
tokenizer1 = Tokenizer(num_words=n_most_common_words1, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer1.fit_on_texts(concated1['review'].values)
sequences1 = tokenizer1.texts_to_sequences(concated1['review'].values)
word_index1 = tokenizer1.word_index
print('Found %s unique tokens.' % len(word_index1))

n_most_common_words2 = 8000
max_len2 = 130
tokenizer2 = Tokenizer(num_words=n_most_common_words2, filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer2.fit_on_texts(concated1['review'].values)
sequences2 = tokenizer2.texts_to_sequences(concated2['review'].values)
word_index2 = tokenizer2.word_index
print('Found %s unique tokens.' % len(word_index2))

X1 = pad_sequences(sequences1, maxlen=max_len1)

X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, labels1, test_size=0.25, random_state
=42)

epochs = 10
emb_dim = 128
batch_size = 256

```

```

labels1[:2]

print((X_train1.shape, y_train1.shape, X_test1.shape, y_test1.shape))

X2 = pad_sequences(sequences2, maxlen=max_len2)

X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, labels2, test_size=0.25, random_state=42)
print((X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape))

forward_layer = LSTM(10, return_sequences=True)
backward_layer = LSTM(10, activation='relu', return_sequences=True, go_backwards=True)
shared=Bidirectional(forward_layer, backward_layer=backward_layer, input_shape=(5, 10))

epochs = 25
model = Sequential()
model.add(Embedding(n_most_common_words1, emb_dim, input_length=X1.shape[1]))
model.add(SpatialDropout1D(0.3))
#model.add(Flatten())
forward_layer = LSTM(10, return_sequences=True)
backward_layer = LSTM(10, activation='relu', return_sequences=True, go_backwards=True)
model.add(Bidirectional(forward_layer, backward_layer=backward_layer, input_shape=(5, 10)))
model.add(shared)
model.add(Flatten())

model.add(Dense(5, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())

history = model.fit(X_train1, y_train1, epochs=45, batch_size=batch_size, validation_split=0.2)

epochs = 25
model1 = Sequential()
model1.add(Embedding(n_most_common_words1, emb_dim, input_length=X1.shape[1]))
model1.add(SpatialDropout1D(0.3))
#model1.add(Flatten())
forward_layer = LSTM(10, return_sequences=True)
backward_layer = LSTM(10, activation='relu', return_sequences=True, go_backwards=True)
model1.add(Bidirectional(forward_layer, backward_layer=backward_layer, input_shape=(5, 10)))
model1.add(shared)
model1.add(Flatten())

model1.add(Dense(5, activation='softmax'))
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print(model1.summary())

history1 = model1.fit(X_train2, y_train2, epochs=45, batch_size=batch_size, validation_split=0.2)

```

```

model2 = Sequential()
model2.add(Embedding(n_most_common_words2, emb_dim, input_length=X2.shape[1]))
model2.add(SpatialDropout1D(0.7))
model2.add(Dense(2, activation='softmax'))
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(model2.summary())

data = pd.read_csv("SUBJ.csv", sep=',', delimiter=',', header='infer', names=None)

data1 = pd.read_csv("IMDB Dataset.csv", sep=',', delimiter=',', header='infer', names=None)

num_of_categories1 = 45000
shuffled1 = data.reindex(np.random.permutation(data.index))
zero1 = shuffled1[shuffled1['sentiment'] == 0][:num_of_categories1]
one1 = shuffled1[shuffled1['sentiment'] == 1][:num_of_categories1]

num_of_categories2 = 45000
shuffled2 = data1.reindex(np.random.permutation(data1.index))
zero2 = shuffled2[shuffled2['sentiment'] == 0][:num_of_categories2]
one2 = shuffled2[shuffled2['sentiment'] == 1][:num_of_categories2]

concated1 = pd.concat([zero1, one1], ignore_index=True)
#Shuffle the dataset
concated1 = concatenated1.reindex(np.random.permutation(concated1.index))
concated1['res1'] = 0

concated2 = pd.concat([one2, zero2], ignore_index=True)
#Shuffle the dataset
concated2 = concatenated2.reindex(np.random.permutation(concated2.index))
concated2['res2'] = 0

concated1.loc[concated1['sentiment'] == 0, 'res1'] = 0
concated1.loc[concated1['sentiment'] == 1, 'res1'] = 1

concated2.loc[concated2['sentiment'] == 0, 'res2'] = 0
concated2.loc[concated2['sentiment'] == 1, 'res2'] = 1

print(concated1['res1'][:10])
labels1 = to_categorical(concated1['res1'], num_classes=2)
print(labels1[:10])
if 'label' in concat1.keys():
    concat1.drop(['label'], axis=1)

print(concated2['res2'][:10])
labels2 = to_categorical(concated2['res2'], num_classes=2)
print(labels2[:10])
if 1 in concat2.keys():

```

```

concated2.drop([1], axis=1)

n_most_common_words1 = 8000
max_len1 = 130
tokenizer1 = Tokenizer(num_words=n_most_common_words1, filters='!"#$%&()*+,-
./:;<=>?@[\\^_`{|}~', lower=True)
tokenizer1.fit_on_texts(concated1['review'].values)
sequences1 = tokenizer1.texts_to_sequences(concated1['review'].values)
word_index1 = tokenizer1.word_index
print('Found %s unique tokens.' % len(word_index1))

n_most_common_words2 = 8000
max_len2 = 130
tokenizer2 = Tokenizer(num_words=n_most_common_words2, filters='!"#$%&()*+,-
./:;<=>?@[\\^_`{|}~', lower=True)
tokenizer2.fit_on_texts(concated1['review'].values)
sequences2 = tokenizer2.texts_to_sequences(concated2['review'].values)
word_index2 = tokenizer2.word_index
print('Found %s unique tokens.' % len(word_index2))

X1 = pad_sequences(sequences1, maxlen=max_len1)

X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, labels1, test_size=0.25, random_state
=42)

epochs = 10
emb_dim = 128
batch_size = 256
labels1[:2]

print((X_train1.shape, y_train1.shape, X_test1.shape, y_test1.shape))

X2 = pad_sequences(sequences2, maxlen=max_len2)

X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, labels2, test_size=0.25, random_state
=42)
print((X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape))

forward_layer = LSTM(10, return_sequences=True)
backward_layer = LSTM(10, activation='relu', return_sequences=True, go_backwards=True)
shared=Bidirectional(forward_layer, backward_layer=backward_layer, input_shape=(5, 10))

epochs = 25
model = Sequential()
model.add(Embedding(n_most_common_words1, emb_dim, input_length=X1.shape[1]))
model.add(SpatialDropout1D(0.3))
#model.add(Flatten())

```



```

forward_layer = LSTM(10, return_sequences=True)
backward_layer = LSTM(10, activation='relu', return_sequences=True, go_backwards=True)
model.add(Bidirectional(forward_layer, backward_layer=backward_layer, input_shape=(5, 10)))
model.add(shared)
model.add(Flatten())

model.add(Dense(2, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model.summary())

history = model.fit(X_train1, y_train1, epochs=45, batch_size=batch_size, validation_split=0.2)

epochs = 25
model1 = Sequential()
model1.add(Embedding(n_most_common_words1, emb_dim, input_length=X1.shape[1]))
model1.add(SpatialDropout1D(0.3))
#model.add(Flatten())
forward_layer = LSTM(10, return_sequences=True)
backward_layer = LSTM(10, activation='relu', return_sequences=True, go_backwards=True)
model1.add(Bidirectional(forward_layer, backward_layer=backward_layer, input_shape=(5, 10)))
model1.add(shared)
model1.add(Flatten())

model1.add(Dense(2, activation='sigmoid'))
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model1.summary())

history1 = model1.fit(X_train2, y_train2, epochs=45, batch_size=batch_size, validation_split=0.2)

model2 = Sequential()
model2.add(Embedding(n_most_common_words2, emb_dim, input_length=X2.shape[1]))
model2.add(SpatialDropout1D(0.7))
model2.add(Dense(2, activation='softmax'))
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model2.summary())

```

CONFIRMATION MAIL:



RMX-CSE PSG CT 11:27 AM

to me ✓



Shaheer,

You can submit this report and presentation for your review.

Regards,

Dr. Marx R, BE (CSE), MS (CS) (USA), PhD,
Associate Professor, Dept. of CSE,
PSG College of Technology, Coimbatore - 641004.