Topic: Sentiment Analysis
MLE501 - Final Project Report

Instructor: Dr. - Nguyen The Anh

Tong Bao Loc, Tran Van Thinh, Nguyen Phu Cuong
loc23mse43006@fsb.edu.vn, thinh23mse43007@fsb.edu.vn,
cuong23mse43010@fsb.edu.vn

July 22, 2024

Contents

Project Description

Sentiment analysis:

- The goal of sentiment analysis is to create tools that automatically understand feelings and opinions from texts, sounds, and images. Analyzing social media content from platforms like Facebook and Twitter requires considering different content types and sharing methods.
- Structured reviews are longer and more coherent compared to short social media posts. Social media posts usually express feelings in a few sentences with informal language and unique spellings.

Project Focus:

- The focus is initially on processing text content, with sound and image analysis upcoming in the next phases.
- The project aims to apply sentiment analysis using the IMDB dataset for experimentation. We will test approaches including Simple RNN, LSTM, and LSTM + Pretrained Word2Vec (Gensim) to classify sentiment into Positive and Negative categories and select the most accurate model possible. The best model will be chosen based on experimental results.

Project Description - Motivation from Biological Neurons

Biological neurons: Biological neurons are composed of a cell body containing the nucleus and most of the cell's complex components, branching extensions called dendrites and a very long extension called axon. There're about 200 billion neurons in our brain, and every of them is connected with about 1000 to 10.000 other neurons via synapses. There are about 125 trillion synapses in human brain. Electrical impulses (signals) flow into the cell through the dendrites, cells are more sensitive to signals through certain dendrites than others. When a cell accumulates signals to a certain threshold, it fires its own signal through the axon.

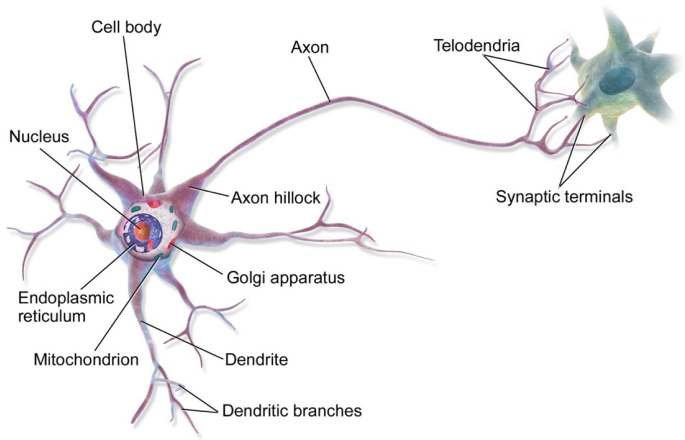


Figure 1: Biological neurons

Project Description - Neuron Network and Deep learning

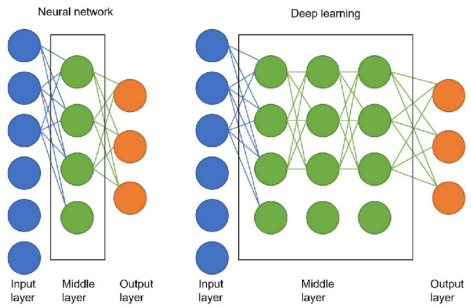


Figure 2: Neuron network and Deep learning Model

	Deep learning systems	Simple neural networks
Architecture	Consists of several hidden layers arranged for convolution or recurrence.	Neural networks consist of an input, hidden, and output layer. They mimic the human brain in structure.
Complexity	Depending on its function, a deep learning network is highly complicated and has structures like long short-term memory (LSTM) and autoencoders.	Neutral networks are less complicated, as they have only a few layers.
Performance	A deep learning algorithm can solve complex issues across large data volumes.	Neural networks perform well when solving simple problems.
Training	It costs a lot of money and resources to train a deep learning algorithm.	The simplicity of a neural network means it costs less to train.

Figure 3: Summary of differences: deep learning systems vs. neural networks

Project Description - Deep Neural Network

There are several deep neural network approaches for Sentiment Analysis.

	Convolutional Neural Network (CNN)	Recurrent Neural Network (RNN)
Intuition	Ability to extract spatial features from input data using self-learning filters	Capable of learning temporal patterns from sequential data
Structure (to this project)	Embedding layer >> Convolution + Pooling >> Dense layer >> Output	Embedding layer >> RNN >> Dense layer >> Output
Speciality	Accuracy in recognizing images	Handling time-series data and sequential information
Data Type	Relies on image data	Sequence data, time-series data
Applications	Image/Video Analytics, Speech Recognition and understanding natural language processing	Time-series forecasting, NLP, Speech Recognition

	Long Short-Term Memory Network (LSTM)	LSTM + Pretrained Word2Vec [Gensim]
Intuition	Capable of learning & remembering patterns across time	Combining long-term memory with pretrained word embeddings for enhanced text processing
Structure (to this project)	Embedding layer >> LSTM >> Dense layer >> Output	Pretrained Word2Vec Embedding >> LSTM >> Dense layer >> Output
Speciality	Memory and self-learning	Leveraging long-term dependencies with semantic knowledge
Data Type	Trained with sequence data	Sequence data, text data with semantic embeddings
Applications	Machine Translation, Language Modeling and Multilingual Language Processing	Text classification, sequence modeling with enhanced memory and semantic understanding

Figure 4: Comparing Solutions

Tasks - Objectives

- Develop a model capable of performing sentiment analysis from text.
- Select a dataset for experimentation.
- Propose model architectures.
- Choose the appropriate frameworks and tools for experimentation.
- Train, test, and evaluate using various metrics.
- Choose the best model to integrate with the application.
- Deploy the model to gather human feedback.

More details: https://github.com/tongbaoloc/sentiment_analysis

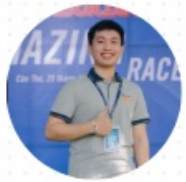
Tasks - Planning

Tasks and schedule

Let's share our accomplishments, tasks, and blockers.

team





Start: Jun 28, 24 End: Jul 15, 24	Thinh Tran	Loc Tong	Cuong Nguyen
Jun 28 to Jul 4	• Introduction • Biological neurons	• Neural networks • Recurrent Neuron Network	• Deep learning • LSTM
Jul 5 to Jul 10	• Objective the project	• Model architecture	• Dataset
Jul 11 to Jul 13	• Framework and tools used	• Experiment and obtained result	• Experiment and obtained result
Jul 14 to Jul 15	• Conclusion and train report	• Conclusion and train report	• Conclusion and train report

Figure 5: Members and tasks

Dataset

Based on the ranking of sentiment datasets from geeksforgeeks.org, we have a high tendency to concentrate datasets which is focuses on a narrow field instead of complex datasets with many layers of meaning and topics from social network platforms like Youtube or Reddit.

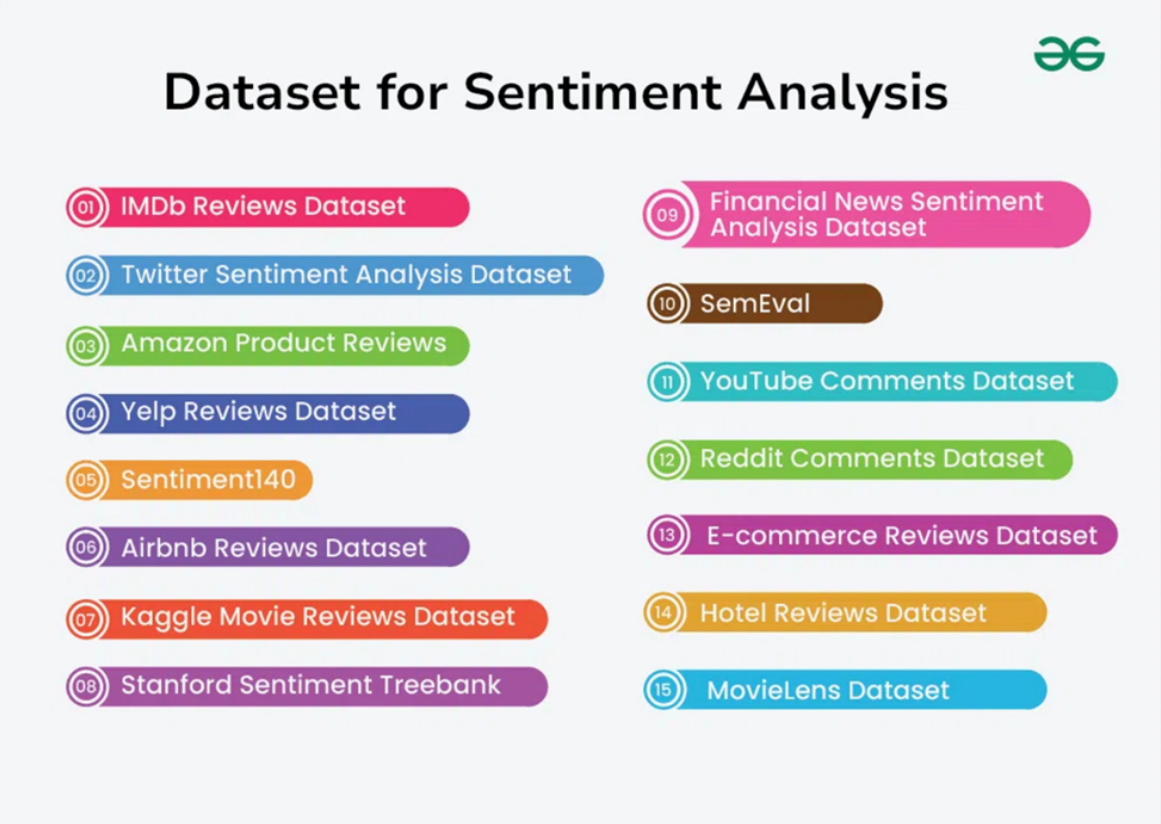


Figure 6: Dataset ranking of sentiment analysis

Dataset

According to the our assessment, the narrowest field in the above ranking is movie review datasets, including 3 datasets:

- IMDb Review Dataset (50k samples): raw data from IMDb collected by Stanford with binary labeled.
- Kaggle Movie Reviews Dataset (2k samples): cleaned data from IMDb with binary labeled.
- Movielens Dataset (20k samples): rating data labeled in many aspects with no linguistically comments.

The topic will use the data set of IMDb Review Dataset because, although the data of Kaggle Movie Reviews Dataset is cleaner, but the number of samples is too small compared to IMDb Review Dataset, and the cleaning process from raw to clean does not take too much effort, so we can handle it our self to ensure the highest quality of data

Dataset

To evaluate IMDb Review Dataset, we use following criteria:

- **Number of entries:** 50000 is the number of records considered good enough when compared to datasets processed by NLP - text classification (from 50000 to 200000 data sample).
- **Completeness:** the dataset has absolutely no loss.
- **Distribution of category variables:** 50000 samples with 25000 negative samples and 25000 positive samples, a perfect distribution can be directly seen without using any calculation such as T-test or chi-square test.
- **Representativeness:** This dataset is taken from IMDb, a reputable and rich data source on movie reviews and taken by Stanford, a large and trustworthy unit in terms of scientific research. This ensures data is collected in proper way with high representation as these reviews come from real users and reflect diverse perspectives on the films.

Model Architecture

This is the high-level architecture proposed for Sentiment Analysis using Neural Network

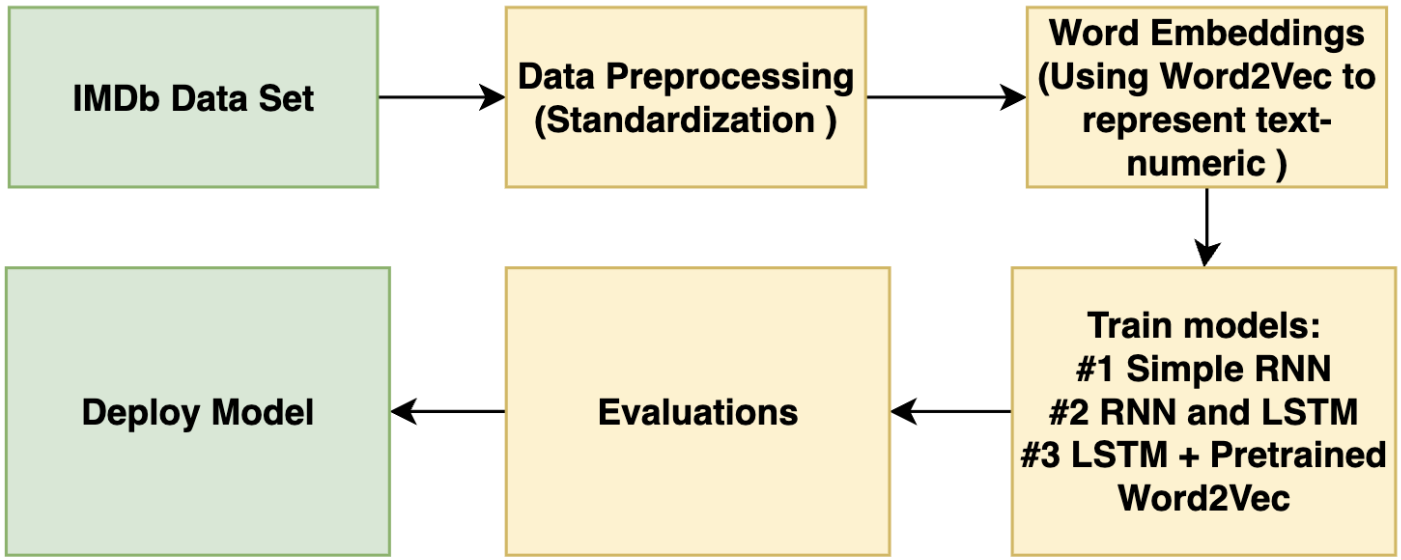


Figure 7: High-level Architecture proposed

Model Architecture - IMDb Data Set - Data Preprocessing

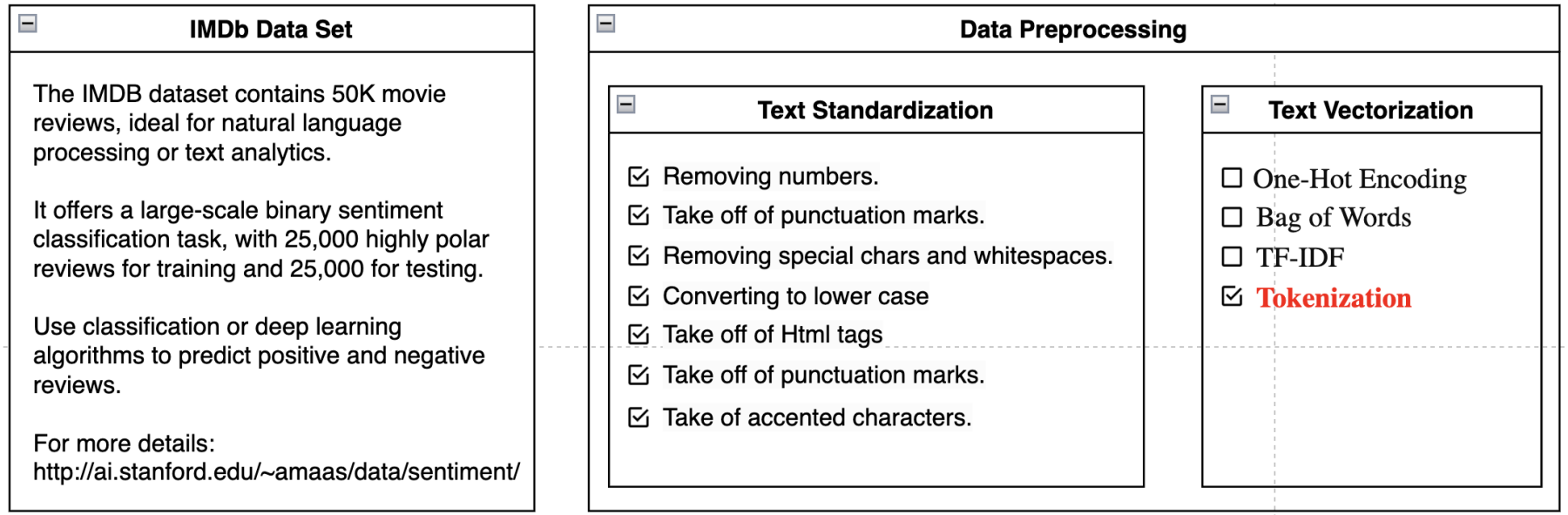


Figure 8: Model Architecture - IMDb Data Set - Data Preprocessing

Model Architecture - Building and deployment

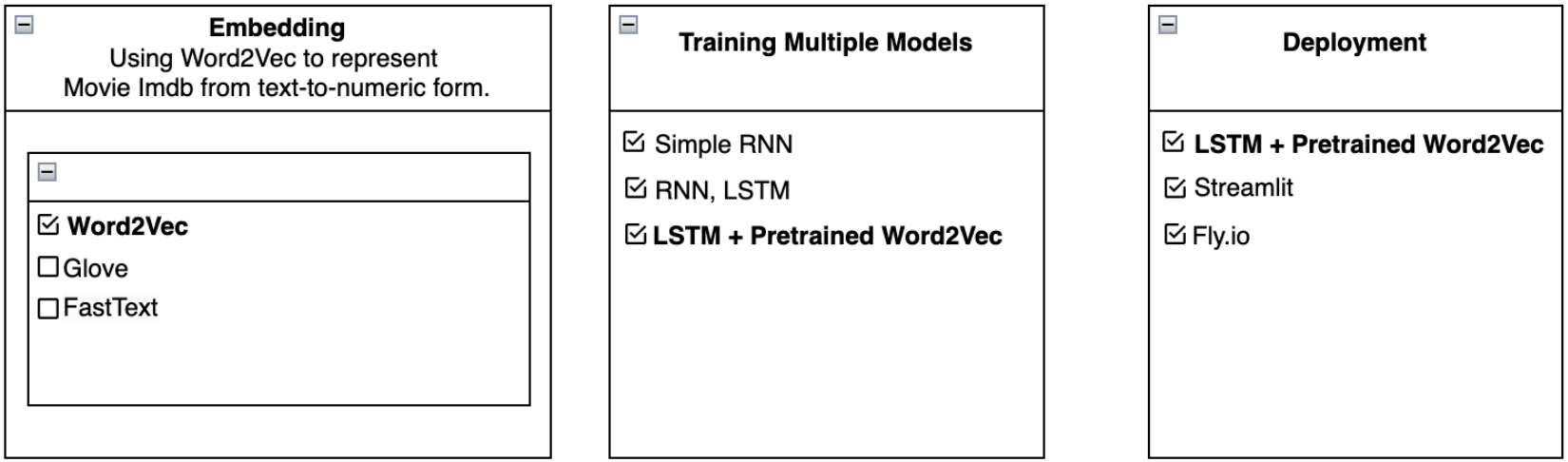


Figure 9: Building and deployment

Deep Learning Development Framework

- **TensorFlow**
  - An open-source platform for machine learning.
  - Provides a comprehensive, flexible ecosystem of tools, libraries, and community resources.
  - Enables researchers to push the state-of-the-art in ML, and developers to easily build and deploy ML-powered applications.
- **Keras**
  - A high-level neural networks API, written in Python.
  - Capable of running on top of TensorFlow, CNTK, or Theano.
  - Allows for easy and fast prototyping, supports both convolutional and recurrent networks, and runs seamlessly on both CPU and GPU.
- **TensorFlow Dataset**
  - A collection of datasets ready-to-use with TensorFlow.
  - Simplifies input pipeline creation using high-level API.
  - Facilitates data augmentation and preprocessing.
- **Streamlit**
  - An open-source app framework for Machine Learning and Data Science projects.
  - Turns data scripts into shareable web apps in minutes.
  - Allows you to present the results of built models interactively and beautifully.

Development Tools

- **Pycharm**: An integrated development environment (IDE) used for programming, especially with Python. It offers code analysis, a graphical debugger, an integrated unit tester, integration with version control systems.
- **Google Colab**: A cloud-based Jupyter notebook environment that allows you to write and execute Python code in your browser, with no configuration required, access to GPUs, and easy sharing.
- **GitHub**: A web-based platform used for version control and collaborative software development. It hosts repositories, tracks changes, and facilitates collaboration through features like pull requests, issues, and wikis.

Deployment Environment

- We used a fully managed host provided by the Fly.io service combined with a GitHub pipeline to automatically deploy the application to the execution environment.
- Access the application at: https://sentiment.locatptn.site

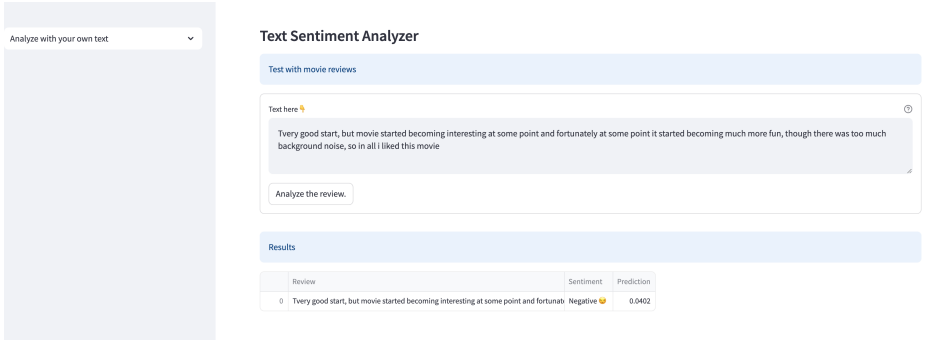


Figure 10: Negative case

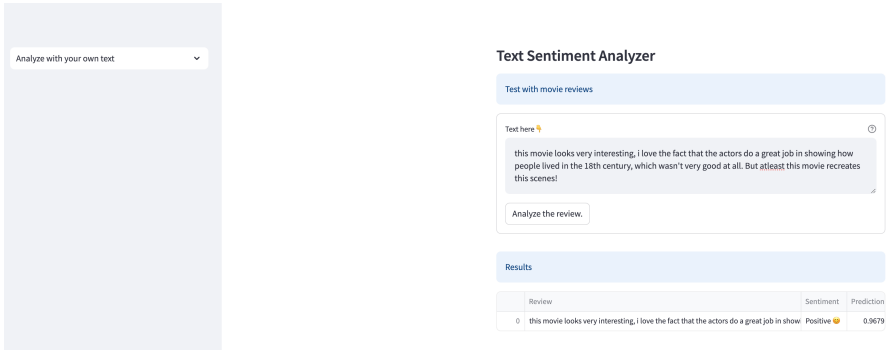


Figure 11: Positive case

Experiment and Obtained Result

- Experiment Setup
  - Data preparation
  - Pretrained Word Embedding model: Word2Vec
- Model Building
  - Simple RNN
  - LSTM
  - LSTM with Pretrained Embedding (Word2Vec)
- Evaluation
- Deployment

Experiment and Obtained Result - Setup

- Data preparation
  - IMDb dataset (49582 records)
  - Split the dataset into training (70%), validation (15%) and test sets (15%)

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Figure 12: Display the first 5 rows of the dataset

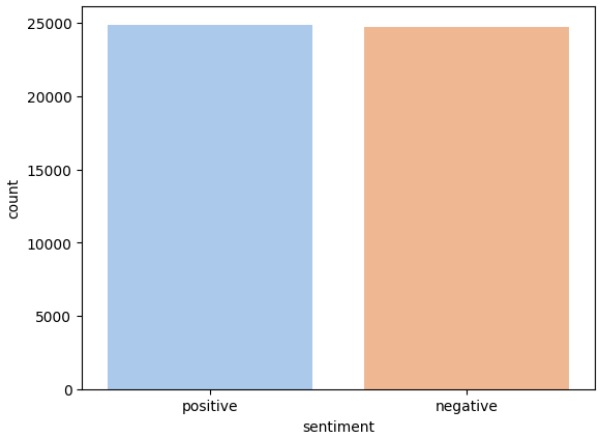


Figure 13: Plot the count of each sentiment

Experiment and Obtained Result - Setup

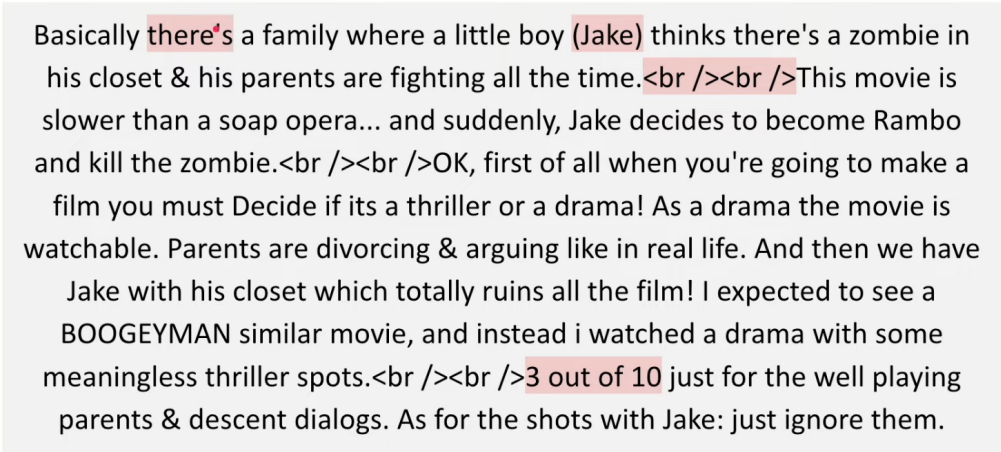


Figure 14:

- Preparing embedding layer (Keras TextVectorization)
  - Standardize data: remove punctuations and numbers, remove HTML tags, single character removal, remove multiple spaces, remove stopwords
  - Hyperparameters: Batch size, Vocabulary, sequence length.
  - Building vocabulary.
  - Tokenization - vectorization (train, validation and test sets).

Experiment and Obtained Result - Setup

- Preparing embedding layer (Keras TextVectorization)

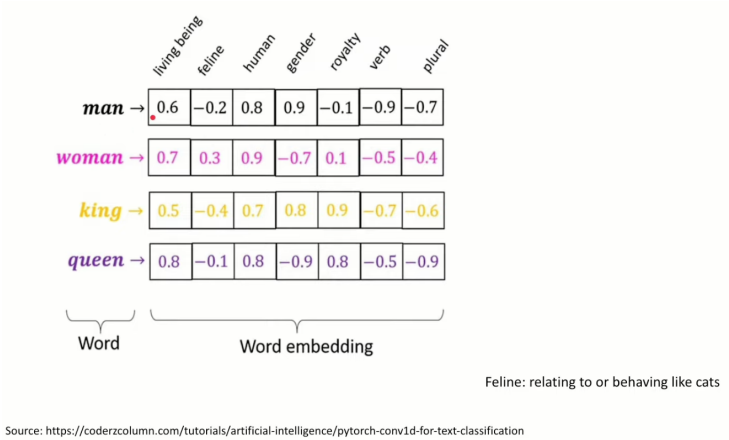
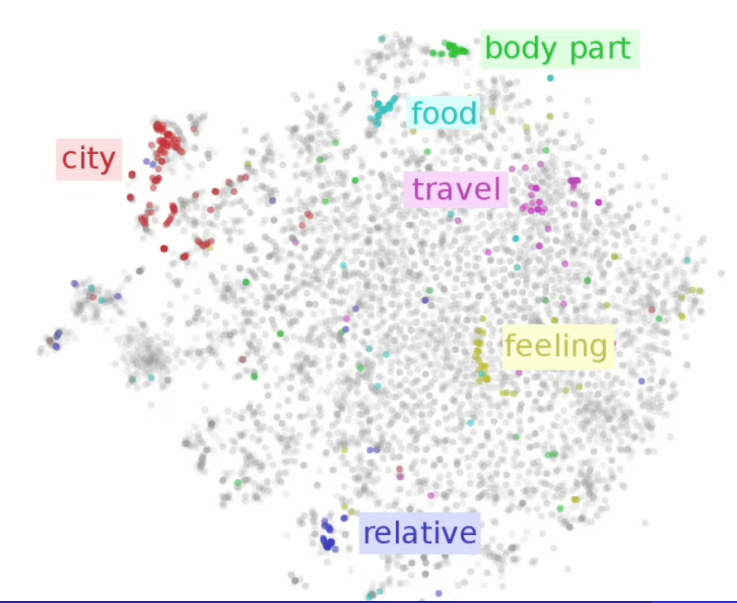


Figure 15: Work embedding



Experiment and Obtained Result - Simple RNN

- The RNN model is well-suited for binary sentiment analysis tasks.
- It leverages an embedding layer for word representation.
- The recurrent layer captures temporal dependencies in the sequence data.
- The final dense layer performs the classification.

Experiment and Obtained Result - Simple RNN Configuration

- **Hyperparameters:**
  - Batch Size: 64
  - Maximum Vocabulary Size: 10,000
  - Maximum Sequence Length: 250
  - Embedding Dimension: 64
  - Number of Epochs: 15
- **Model Architecture:**
  - Sequential Model
  - Input Layer: shape = (250,)
  - Embedding Layer: vocab_size = 10,000, embedding_dim = 64
  - SimpleRNN Layer: units = 32
  - Dense Layer: units = 1, activation = 'sigmoid'
- **Model Summary:**
  - Embedding Layer: Output Shape = (None, 250, 64), Params = 6,400,000
  - SimpleRNN Layer: Output Shape = (None, 32), Params = 3,104
  - Dense Layer: Output Shape = (None, 1), Params = 33
- **Model Parameters:**
  - Total params: 6,431,37 (2.45 MB)
  - Trainable params: 6,431,37 (2.45 MB)
  - Non-trainable params: 0 (0.00 Byte)

Experiment and Obtained Result - Simple RNN

- Output log

```
Training the model...
Epoch 1/15
543/543 [==============================] - 20s 35ms/step - loss: 0.6940 - accuracy: 0.5144 - val_loss: 0.6939 - val_accuracy: 0.5026
Epoch 2/15
543/543 [==============================] - 18s 34ms/step - loss: 0.6857 - accuracy: 0.5428 - val_loss: 0.6929 - val_accuracy: 0.5338
Epoch 3/15
543/543 [==============================] - 18s 34ms/step - loss: 0.6755 - accuracy: 0.5757 - val_loss: 0.6940 - val_accuracy: 0.5206
Epoch 4/15
543/543 [==============================] - 18s 34ms/step - loss: 0.5585 - accuracy: 0.7433 - val_loss: 0.4881 - val_accuracy: 0.7974
Epoch 5/15
543/543 [==============================] - 19s 34ms/step - loss: 0.3769 - accuracy: 0.8630 - val_loss: 0.4125 - val_accuracy: 0.8415
Epoch 6/15
543/543 [==============================] - 19s 35ms/step - loss: 0.2921 - accuracy: 0.9021 - val_loss: 0.3599 - val_accuracy: 0.8603
Epoch 7/15
543/543 [==============================] - 19s 34ms/step - loss: 0.2415 - accuracy: 0.9231 - val_loss: 0.3758 - val_accuracy: 0.8390
Epoch 8/15
543/543 [==============================] - 19s 35ms/step - loss: 0.2107 - accuracy: 0.9362 - val_loss: 0.3690 - val_accuracy: 0.8331
Epoch 9/15
543/543 [==============================] - 20s 36ms/step - loss: 0.1831 - accuracy: 0.9466 - val_loss: 0.3504 - val_accuracy: 0.8581
Epoch 10/15
543/543 [==============================] - 19s 36ms/step - loss: 0.1685 - accuracy: 0.9521 - val_loss: 0.3769 - val_accuracy: 0.8658
Epoch 11/15
543/543 [==============================] - 18s 33ms/step - loss: 0.1517 - accuracy: 0.9574 - val_loss: 0.4081 - val_accuracy: 0.8550
Epoch 12/15
543/543 [==============================] - 18s 34ms/step - loss: 0.1344 - accuracy: 0.9635 - val_loss: 0.3980 - val_accuracy: 0.8618
Epoch 13/15
543/543 [==============================] - 18s 33ms/step - loss: 0.1185 - accuracy: 0.9694 - val_loss: 0.4513 - val_accuracy: 0.8599
Epoch 14/15
543/543 [==============================] - 18s 33ms/step - loss: 0.1063 - accuracy: 0.9733 - val_loss: 0.4823 - val_accuracy: 0.8516
Epoch 15/15
543/543 [==============================] - 18s 34ms/step - loss: 0.1025 - accuracy: 0.9746 - val_loss: 0.4479 - val_accuracy: 0.8592
Time taken to train the model:  279.7931318283081
```

Figure 17: RNN Output log

Experiment and Obtained Result - Simple RNN

- Plotting model loss and accuracy

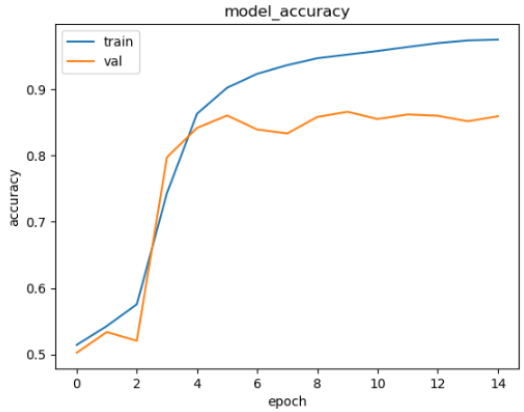


Figure 18: Accuracy

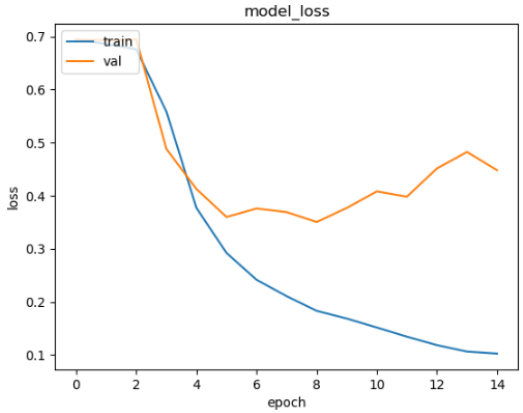


Figure 19: Loss

Experiment and Obtained Result - Simple RNN

- Training Results Summary
  - Accuracy
    - Training accuracy improves consistently from 0.8687 to 0.9584.
    - Validation accuracy fluctuates around 0.78 to 0.80.
  - Loss
    - Training loss decreases from 0.3370 to 0.1347.
    - Validation loss fluctuates and generally increases, indicating potential overfitting.
- Evaluate with test data (unseen)

```
117/117 [==============================] - 2s 14ms/step - loss: 0.4300 - accuracy: 0.8629
[0.42998144030570984, 0.8628663420677185]
```

Figure 20: Evaluate with unseen data

Experiment and Obtained Result - Simple RNN

- Ideals for Improvement
  - To improve the model's performance, especially on the validation set, consider using an LSTM (Long Short-Term Memory) network. LSTMs can capture long-range dependencies better than Simple RNNs, which may help reduce overfitting and improve generalization.
  - **Use of LSTM Layers**: Replace the SimpleRNN layers with LSTM layers to better capture long-range dependencies.
  - **Bidirectional LSTM**: Use bidirectional LSTMs to provide the model with information from both past and future states.
  - **Dropout Regularization**: Introduce dropout layers to prevent overfitting by randomly dropping units during training.
  - **Batch Normalization**: Apply batch normalization to stabilize and speed up the training process.
  - **Early Stopping**: Implement early stopping to terminate training when validation performance stops improving.
  - **Learning Rate Scheduler**: Adjust the learning rate dynamically based on the training progress.

Experiment and Obtained Result - LSTM

- **Handling Long-Term Dependencies**:
  - LSTMs can remember information for long periods, making them suitable for analyzing long texts where the sentiment may depend on information from earlier in the sequence.
- **Sequence Modeling**:
  - LSTMs process input sequences sequentially, maintaining an internal state that captures contextual information, which is crucial for understanding the sentiment in phrases and sentences.
- **Contextual Understanding**:
  - LSTMs can capture the context in which words are used, differentiating between positive and negative sentiment depending on context (e.g., "not good" vs. "good").
- **Bidirectional LSTM**:
  - Using bidirectional LSTMs enhances sentiment analysis by allowing the model to understand context from both past and future states, leading to better accuracy.
- **Reducing Overfitting with Dropout**:
  - Dropout regularization can be applied in LSTM networks to prevent overfitting, improving the model's ability to generalize to unseen data.
- **Robust to Variations in Input Length**:
  - LSTMs can handle input sequences of varying lengths, making them flexible

Experiment and Obtained Result - LSTM Model Configuration

Hyperparameters

- **Batch Size**: 64
- **Maximum Vocabulary Size**: 10,000
- **Maximum Sequence Length**: 250
- **Embedding Dimension**: 64
- **Number of Epochs**: 15

Model Architecture

- **Sequential Model**
- **Input Layer**: shape = (250,)
- **Embedding Layer**: vocab size = 10,000, embedding dim = 64
- **Bidirectional LSTM Layer**: units = 64, return sequences = True
- **Dropout Layer**: rate = 0.7
- **Bidirectional LSTM Layer**: units = 32
- **Dense Layer**: units = 64, activation = 'relu'
- **Dropout Layer**: rate = 0.7
- **Dense Layer with L2 Regularization**: units = 1, activation = 'sigmoid', regularization = L2(0.01)

Experiment and Obtained Result - LSTM

- Output log

```
Training the model...
Epoch 1/15
543/543 [==============================] - 158s 282ms/step - loss: 0.6610 - accuracy: 0.5892 - val_loss: 0.4271 - val_accuracy: 0.8161
Epoch 2/15
543/543 [==============================] - 147s 271ms/step - loss: 0.3510 - accuracy: 0.8689 - val_loss: 0.3222 - val_accuracy: 0.8735
Epoch 3/15
543/543 [==============================] - 147s 270ms/step - loss: 0.2648 - accuracy: 0.9105 - val_loss: 0.3374 - val_accuracy: 0.8731
Epoch 4/15
543/543 [==============================] - 141s 260ms/step - loss: 0.2245 - accuracy: 0.9273 - val_loss: 0.3461 - val_accuracy: 0.8775
Time taken to train the model:  593.0148024559021
```

Figure 21: LSTM Output log

Experiment and Obtained Result - LSTM

- Plotting model loss and accuracy

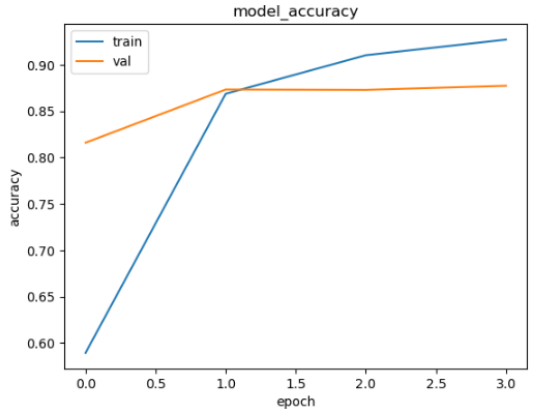


Figure 22: Accuracy

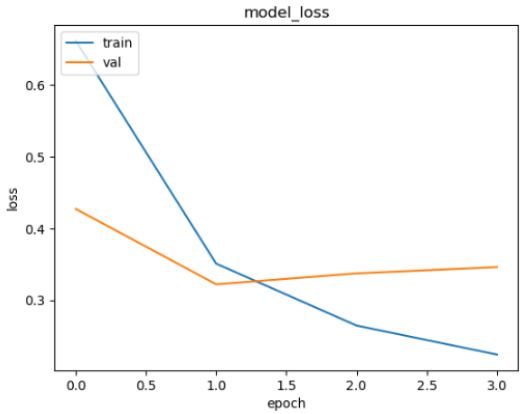


Figure 23: Loss

Experiment and Obtained Result - LSTM

Training Results Summary

- Accuracy
  - Training accuracy improves consistently from 0.9790 to 0.9826.
  - Validation accuracy fluctuates around 0.8663 to 0.8609.
- Loss
  - Training loss decreases from 0.0905 to 0.0785.
  - Validation loss fluctuates and generally increases, indicating potential overfitting.

Evaluate with test data (unseen)

```
117/117 [==============================] - 8s 69ms/step - loss: 0.3151 - accuracy: 0.8771
: [0.31505218148231506, 0.8771175146102905]
```

Figure 24: Evaluate with unseen data

Experiment and Obtained Results - LSTM

- Ideas for Improvement
  - Leverage Pre-trained Embedding Models (e.g., Pre-trained Word2Vec)

Experiment and Obtained Result - LSTM + Pre-trained Embedding

- **Handling Long-Term Dependencies**:
  - Word2Vec is a popular technique developed by Google for natural language processing tasks. It is a two-layer neural network that processes text by converting words into vectors of numbers. These vectors are created in such a way that words with similar meanings end up having similar vector representations. This technique has significantly advanced the field of NLP by enabling models to understand and work with the semantics of words.

Experiment and Obtained Result - LSTM + Pre-trained Embedding Model Configuration

Hyperparameters

- **Batch Size**: 64
- **Maximum Vocabulary Size**: 10,000
- **Maximum Sequence Length**: 250
- **Embedding Dimension**: pretrained_embeddings_array.shape[1]
- **Number of Epochs**: 15

Model Architecture

- **Sequential Model**
- **Input Layer**: shape = (250,)
- **Embedding Layer**: vocab size = 10,000, embedding dim = pretrained_embeddings_array.shape[1] , pretrained embeddings, trainable = True
- **Bidirectional LSTM Layer**: units = 64, return sequences = True
- **Bidirectional LSTM Layer**: units = 32
- **Dense Layer**: units = 32, activation = 'relu'
- **Dropout Layer**: rate = 0.5
- **Dense Layer**: units = 1, activation = 'sigmoid'

Experiment and Obtained Result - LSTM + Pre-trained Embedding

- Output log

```
Training the model...
Epoch 1/10
543/543 [==============================] - 176s 315ms/step - loss: 6.2669 - accuracy: 0.5181 - val_loss: 4.1963 - val_accuracy: 0.6371
Epoch 2/10
543/543 [==============================] - 174s 320ms/step - loss: 3.2308 - accuracy: 0.6703 - val_loss: 2.3905 - val_accuracy: 0.7939
Epoch 3/10
543/543 [==============================] - 174s 321ms/step - loss: 1.9643 - accuracy: 0.8301 - val_loss: 1.6536 - val_accuracy: 0.8217
Epoch 4/10
543/543 [==============================] - 174s 320ms/step - loss: 1.3652 - accuracy: 0.8689 - val_loss: 1.1933 - val_accuracy: 0.8482
Epoch 5/10
543/543 [==============================] - 171s 316ms/step - loss: 1.0070 - accuracy: 0.8894 - val_loss: 0.9240 - val_accuracy: 0.8768
Epoch 6/10
543/543 [==============================] - 173s 319ms/step - loss: 0.7789 - accuracy: 0.9041 - val_loss: 0.7456 - val_accuracy: 0.8797
Epoch 7/10
543/543 [==============================] - 170s 313ms/step - loss: 0.6334 - accuracy: 0.9130 - val_loss: 0.6720 - val_accuracy: 0.8724
Epoch 8/10
543/543 [==============================] - 170s 314ms/step - loss: 0.5319 - accuracy: 0.9234 - val_loss: 0.6444 - val_accuracy: 0.8661
Epoch 9/10
543/543 [==============================] - 170s 312ms/step - loss: 0.4611 - accuracy: 0.9310 - val_loss: 0.6045 - val_accuracy: 0.8661
Epoch 10/10
543/543 [==============================] - 170s 313ms/step - loss: 0.4059 - accuracy: 0.9374 - val_loss: 0.5790 - val_accuracy: 0.8713
Time taken to train the model:  1723.5508306026459
```

Figure 25: Output log

Experiment and Obtained Result - LSTM + Pre-trained Embedding

- Plotting model loss and accuracy

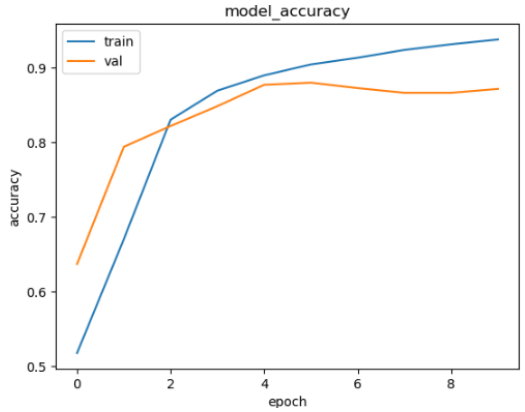


Figure 26: Accuracy

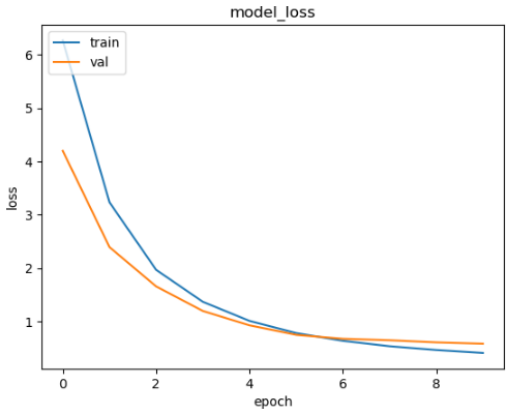


Figure 27: Loss

Experiment and Obtained Result - LSTM + Pre-trained Embedding

Training Results Summary

- Accuracy
  - Training accuracy improves consistently from 0.5518 to 0.2320.
  - Validation accuracy fluctuates around 0.8457 to 0.8849.
- Loss
  - Training loss decreases from 0.2054 to 0.3992.
  - Reduce overfitting compare with below models,
  - The modifications made to the model, including adding dropout, batch normalization, and L2 regularization, seem to have effectively reduced overfitting. The validation performance is closely following the training performance, indicating a well-generalized model.

Evaluate with test data (unseen)

```
117/117 [==============================] - 10s 85ms/step - loss: 0.5819 - accuracy: 0.8734
 [0.5818579196929932, 0.8733530640602112]
```

Figure 28: Evaluate with unseen data

Experiment and Obtained Results - LSTM

- Ideas for Improvement
  - Increase Dropout: If you still observe slight overfitting, consider increasing the dropout rate slightly.
  - Hyperparameter Tuning: Experiment with different hyperparameters (e.g., learning rate, number of LSTM units) to see if you can achieve even better performance.
  - More Data: If possible, training on more data can help improve the model's performance further.

Conclusion

After experimentally evaluating the three models—SimpleRNN, LSTM, and LSTM+Pretrained Embedding—we have chosen to deploy the LSTM+Pretrained Embedding model. This decision is based on its current performance and the lower degree of overfitting compared to the other models. For future work, we plan to implement the following improvements to enhance the model quality:

- **Hyperparameter Tuning:**Conduct a thorough hyperparameter search to identify the optimal settings for learning rate, batch size, number of LSTM units, and dropout rates.
- **Data Augmentation:**Explore techniques to augment the training data, potentially including synthetic data generation or noise addition, to increase the diversity and robustness of the training set.
- **Transfer Learning:**Utilize transfer learning by fine-tuning pre-trained models on related tasks to leverage existing knowledge and improve performance.
- **Regularization Techniques:**Investigate additional regularization techniques, such as weight pruning or variational dropout, to further reduce overfitting.
- **Increased Training Data:**Collect and incorporate more training data to improve the model's ability to generalize to unseen examples.

References

1. Sentiment Analysis with LSTM — Deep Learning with Keras — Neural Networks — Project8 - YouTube
2. Ng, A. (n.d.). Neural Networks and Deep Learning. Coursera. Retrieved from https://www.coursera.org/learn/neural-networks-deep-learning
3. Ng, A. (n.d.). Sequence Models. Coursera. Retrieved from https://www.coursera.org/learn/sequence-models
4. Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781. Retrieved from https://arxiv.org/abs/1301.3781
5. Hochreiter, S., Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. Retrieved from https://www.bioinf.jku.at/publications/older/2604.pdf
6. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11), 2278-2324. Retrieved from http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Discussions

Thank you for your attentions and feedbacks.