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 13, 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 - Brief History of Deep Learning

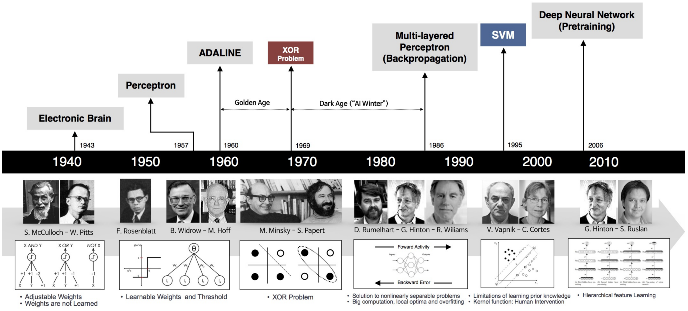

Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Figure 1: Deep learning History

Perceptrons (1950s – 1960s)
Multilayer Perceptron (MLP) and Backpropagation (1980s)
Second AI winter (1990s-2000s)
Deep learning (2006)
The breakthrough (2012)

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.

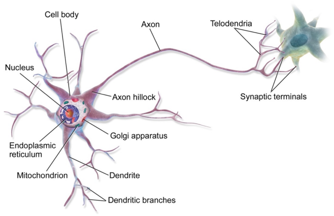
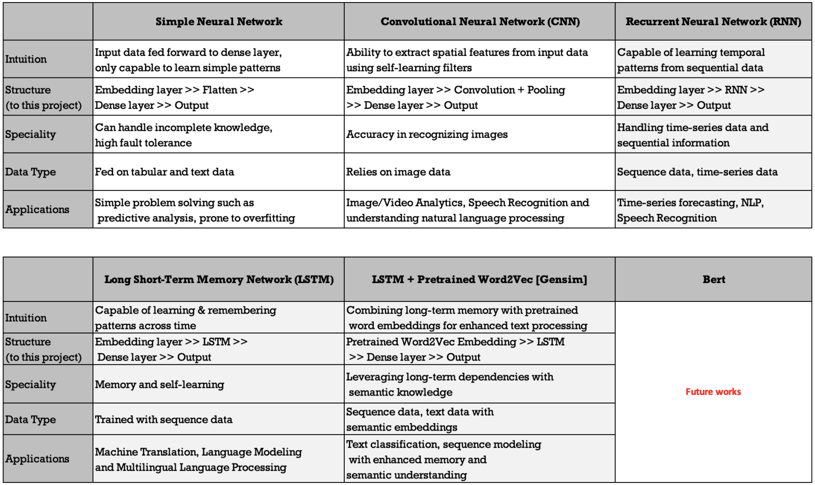


Figure 2: Biological neurons

Project Description - Deep Neural Network

There are several deep neural network approaches for Sentiment Analysis.



	Simple Neural Network	Convolutional Neural Network (CNN)	Recurrent Neural Network (RNN)
Intuition	Input data fed forward to dense layer, only capable to learn simple patterns	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 >> Flatten >> Dense layer >> Output	Embedding layer >> Convolution + Pooling >> Dense layer >> Output	Embedding layer >> RNN >> Dense layer >> Output
Speciality	Can handle incomplete knowledge, high fault tolerance	Accuracy in recognizing images	Handling time-series data and sequential information
Data Type	Fed on tabular and text data	Relies on image data	Sequence data, time-series data
Applications	Simple problem solving such as predictive analysis, prone to overfitting	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]	Bert
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	Future works
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	

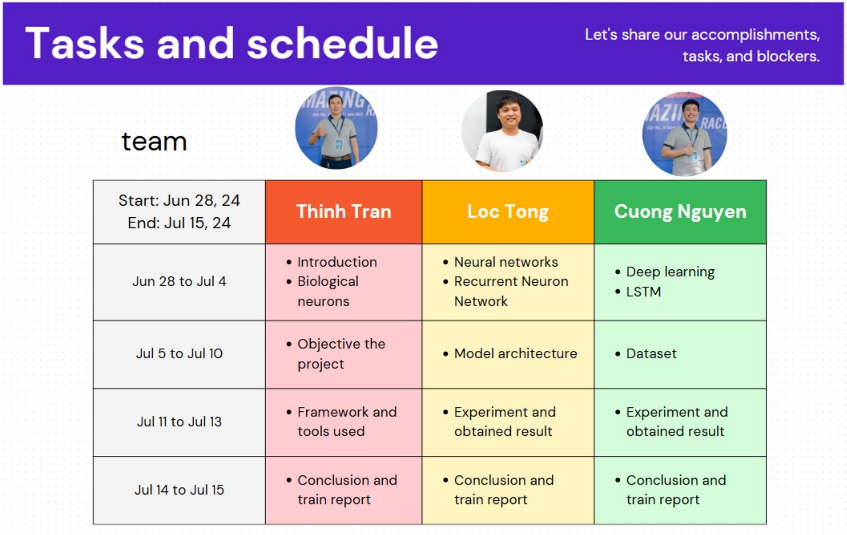
Figure 3: Comparing Solutions

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.

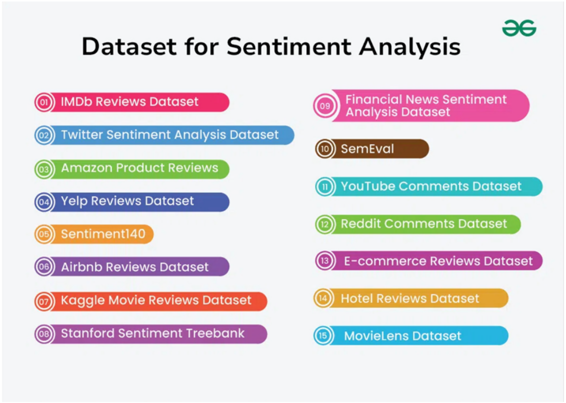
team

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 4: 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.


Dataset for Sentiment Analysis

01 IMDb Reviews Dataset
02 Twitter Sentiment Analysis Dataset
03 Amazon Product Reviews
04 Yelp Reviews Dataset
05 Sentiment140
06 Airbnb Reviews Dataset
07 Kaggle Movie Reviews Dataset
08 Stanford Sentiment Treebank
09 Financial News Sentiment Analysis Dataset
10 SemEval
11 YouTube Comments Dataset
12 Reddit Comments Dataset
13 E-commerce Reviews Dataset
14 Hotel Reviews Dataset
15 MovieLens Dataset

Figure 5: 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 Standford 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

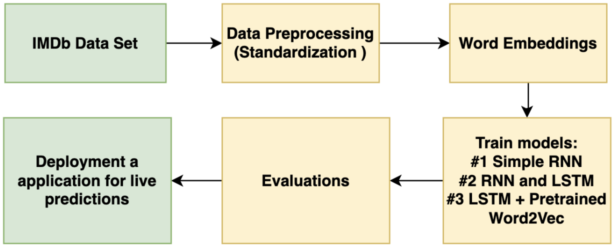
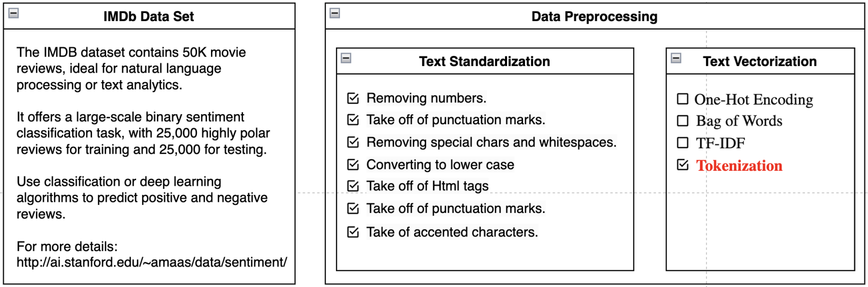

IMDb Data Set
Data Preprocessing (Standardization)
Word Embeddings
Train models: #1 Simple RNN #2 RNN and LSTM #3 LSTM + Pretrained Word2Vec
Evaluations
Deployment a application for live predictions

Figure 6: High-level Architecture proposed

Model Architecture - IMDb Data Set - Data Preprocessing



IMDb Data Set

The IMDB dataset contains 50K movie reviews, ideal for natural language processing or text analytics.

It offers a large-scale binary sentiment classification task, with 25,000 highly polar reviews for training and 25,000 for testing.

Use classification or deep learning algorithms to predict positive and negative reviews.

For more details:
http://ai.stanford.edu/~amaas/data/sentiment/

Data Preprocessing

Text Standardization

- ☑ Removing numbers.
- ☑ Take off of punctuation marks.
- ☑ Removing special chars and whitespaces.
- ☑ Converting to lower case
- ☑ Take off of Html tags
- ☑ Take off of punctuation marks.
- ☑ Take of accented characters.

Text Vectorization

- ☐ One-Hot Encoding
- ☐ Bag of Words
- ☐ TF-IDF
- ☑ **Tokenization**

Figure 7: Model Architecture - IMDb Data Set - Data Preprocessing

Model Architecture - Building and deployment

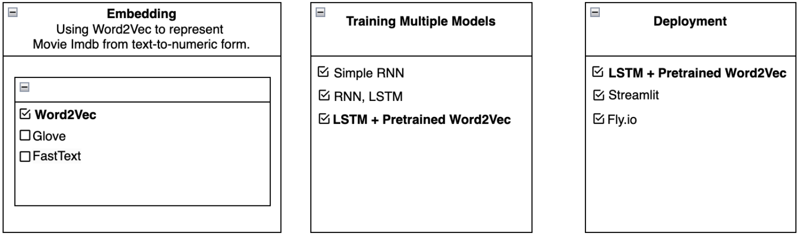

Embedding
Using Word2Vec to represent
Movie Imdb from text-to-numeric form.
☑ Word2Vec
☐ Glove
☐ FastText
Training Multiple Models
☑ Simple RNN
☑ RNN, LSTM
☑ LSTM + Pretrained Word2Vec
Deployment
☑ LSTM + Pretrained Word2Vec
☑ Streamlit
☑ Fly.io

Figure 8: Building and deployment

Deep Learning Development Framework

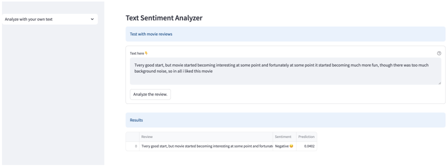
- **TensorFlow**
  - An open-source platform for machine learning.
  - Provides a comprehensive, flexible ecosystem of tools, libraries, and community resources.
  - Enables researchers to push the state-of-the-art in ML, and developers to easily build and deploy ML-powered applications.
- **Keras**
  - 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

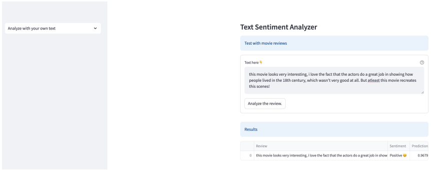
- We used a fully managed host provided by the Fly.io service combined with a GitHub pipeline to automatically deploy the application to the execution environment.
- Access the application at: https://sentiment.locatptn.site



Tvery good start, but movie started becoming interesting at some point and fortunately at some point it started becoming much more fun, though there was too much background noise, so in all i liked this movie

	Review	Sentiment	Prediction
0	Tvery good start, but movie started becoming interesting at some point and fortunat	Negative 😞	0.0402

Figure 9: Negative case



this movie looks very interesting, i love the fact that the actors do a great job in showing how people lived in the 18th century, which wasn't very good at all. But atleast this movie recreates this scenes!

	Review	Sentiment	Prediction
0	this movie looks very interesting, i love the fact that the actors do a great job in show	Positive 😊	0.9679

Figure 10: 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 (50,000 records)
  - Split the dataset into training (70%), validation (15%) and test sets (15%)
  - 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 - Simple RNN

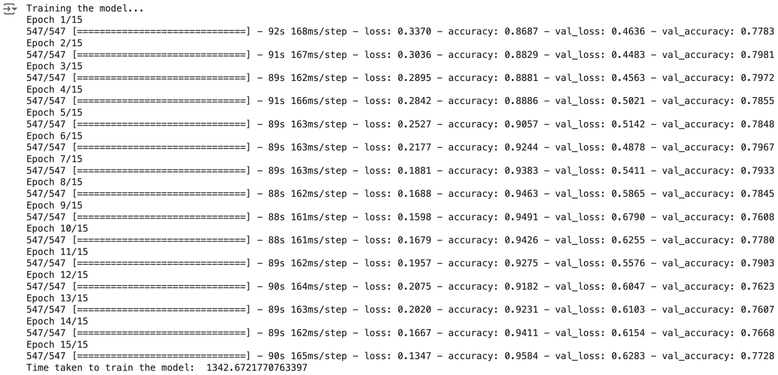
- The RNN model is well-suited for binary sentiment analysis tasks.
- It leverages an embedding layer for word representation.
- The recurrent layer captures temporal dependencies in the sequence data.
- The final dense layer performs the classification.

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
547/547 [==============================] - 92s 168ms/step - loss: 0.3370 - accuracy: 0.8687 - val_loss: 0.4636 - val_accuracy: 0.7783
Epoch 2/15
547/547 [==============================] - 91s 167ms/step - loss: 0.3036 - accuracy: 0.8829 - val_loss: 0.4483 - val_accuracy: 0.7981
Epoch 3/15
547/547 [==============================] - 89s 162ms/step - loss: 0.2895 - accuracy: 0.8881 - val_loss: 0.4563 - val_accuracy: 0.7972
Epoch 4/15
547/547 [==============================] - 91s 166ms/step - loss: 0.2842 - accuracy: 0.8886 - val_loss: 0.5021 - val_accuracy: 0.7855
Epoch 5/15
547/547 [==============================] - 89s 163ms/step - loss: 0.2527 - accuracy: 0.9057 - val_loss: 0.5142 - val_accuracy: 0.7848
Epoch 6/15
547/547 [==============================] - 89s 163ms/step - loss: 0.2177 - accuracy: 0.9244 - val_loss: 0.4878 - val_accuracy: 0.7967
Epoch 7/15
547/547 [==============================] - 89s 163ms/step - loss: 0.1881 - accuracy: 0.9383 - val_loss: 0.5411 - val_accuracy: 0.7933
Epoch 8/15
547/547 [==============================] - 88s 162ms/step - loss: 0.1688 - accuracy: 0.9463 - val_loss: 0.5865 - val_accuracy: 0.7845
Epoch 9/15
547/547 [==============================] - 88s 161ms/step - loss: 0.1598 - accuracy: 0.9491 - val_loss: 0.6790 - val_accuracy: 0.7608
Epoch 10/15
547/547 [==============================] - 88s 161ms/step - loss: 0.1679 - accuracy: 0.9426 - val_loss: 0.6255 - val_accuracy: 0.7780
Epoch 11/15
547/547 [==============================] - 89s 162ms/step - loss: 0.1957 - accuracy: 0.9275 - val_loss: 0.5576 - val_accuracy: 0.7903
Epoch 12/15
547/547 [==============================] - 90s 164ms/step - loss: 0.2075 - accuracy: 0.9182 - val_loss: 0.6047 - val_accuracy: 0.7623
Epoch 13/15
547/547 [==============================] - 89s 163ms/step - loss: 0.2020 - accuracy: 0.9231 - val_loss: 0.6103 - val_accuracy: 0.7607
Epoch 14/15
547/547 [==============================] - 89s 162ms/step - loss: 0.1667 - accuracy: 0.9411 - val_loss: 0.6154 - val_accuracy: 0.7668
Epoch 15/15
547/547 [==============================] - 90s 165ms/step - loss: 0.1347 - accuracy: 0.9584 - val_loss: 0.6283 - val_accuracy: 0.7728
Time taken to train the model:  1342.6721770763397
```

Figure 11: RNN Output log

Experiment and Obtained Result - Simple RNN

- Plotting model loss and accuracy

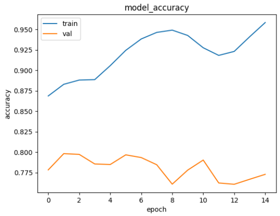


Figure 12: Accuracy

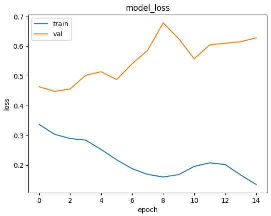
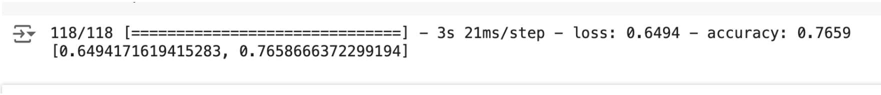


Figure 13: Loss

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)



```
118/118 [==============================] - 3s 21ms/step - loss: 0.6494 - accuracy: 0.7659
[0.6494171619415283, 0.7658666372299194]
```

Figure 14: 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

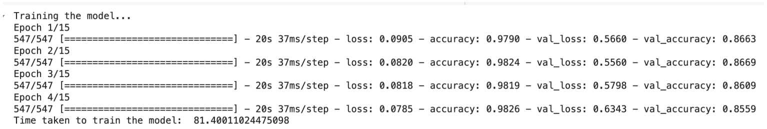
- **Batch Size**: 64
- **Maximum Vocabulary Size**: 10,000
- **Maximum Sequence Length**: 250
- **Embedding Dimension**: 64
- **Number of Epochs**: 15

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
547/547 [==============================] - 20s 37ms/step - loss: 0.0905 - accuracy: 0.9790 - val_loss: 0.5660 - val_accuracy: 0.8663
Epoch 2/15
547/547 [==============================] - 20s 37ms/step - loss: 0.0820 - accuracy: 0.9824 - val_loss: 0.5560 - val_accuracy: 0.8669
Epoch 3/15
547/547 [==============================] - 20s 37ms/step - loss: 0.0818 - accuracy: 0.9819 - val_loss: 0.5798 - val_accuracy: 0.8609
Epoch 4/15
547/547 [==============================] - 20s 37ms/step - loss: 0.0785 - accuracy: 0.9826 - val_loss: 0.6343 - val_accuracy: 0.8559
Time taken to train the model:  81.40011024475098
```

Figure 15: LSTM Output log

Experiment and Obtained Result - LSTM

- Plotting model loss and accuracy

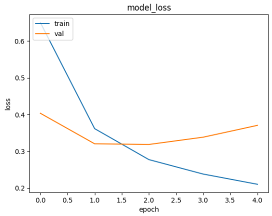


Figure 16: Accuracy

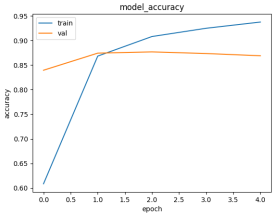


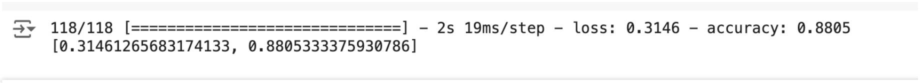
Figure 17: Loss

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)



```
118/118 [==============================] - 2s 19ms/step - loss: 0.3146 - accuracy: 0.8805
[0.31461265683174133, 0.8805333375930786]
```

Figure 18: 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

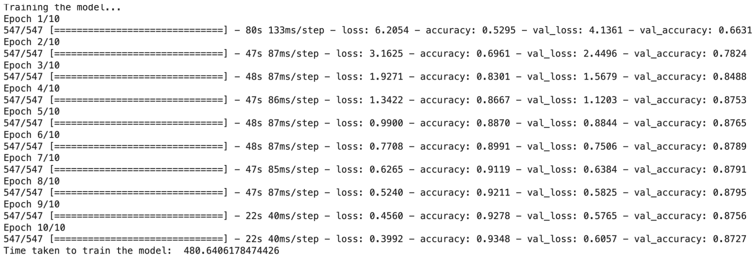
- **Batch Size**: 64
- **Maximum Vocabulary Size**: 10,000
- **Maximum Sequence Length**: 250
- **Embedding Dimension**: pretrained_embeddings_array.shape[1]
- **Number of Epochs**: 15

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
547/547 [==============================] - 80s 133ms/step - loss: 6.2054 - accuracy: 0.5295 - val_loss: 4.1361 - val_accuracy: 0.6631
Epoch 2/10
547/547 [==============================] - 47s 87ms/step - loss: 3.1625 - accuracy: 0.6961 - val_loss: 2.4496 - val_accuracy: 0.7824
Epoch 3/10
547/547 [==============================] - 48s 87ms/step - loss: 1.9271 - accuracy: 0.8301 - val_loss: 1.5679 - val_accuracy: 0.8488
Epoch 4/10
547/547 [==============================] - 47s 86ms/step - loss: 1.3422 - accuracy: 0.8667 - val_loss: 1.1203 - val_accuracy: 0.8753
Epoch 5/10
547/547 [==============================] - 48s 87ms/step - loss: 0.9900 - accuracy: 0.8870 - val_loss: 0.8844 - val_accuracy: 0.8765
Epoch 6/10
547/547 [==============================] - 48s 87ms/step - loss: 0.7708 - accuracy: 0.8991 - val_loss: 0.7506 - val_accuracy: 0.8789
Epoch 7/10
547/547 [==============================] - 47s 85ms/step - loss: 0.6265 - accuracy: 0.9119 - val_loss: 0.6384 - val_accuracy: 0.8791
Epoch 8/10
547/547 [==============================] - 47s 87ms/step - loss: 0.5240 - accuracy: 0.9211 - val_loss: 0.5825 - val_accuracy: 0.8795
Epoch 9/10
547/547 [==============================] - 22s 40ms/step - loss: 0.4560 - accuracy: 0.9278 - val_loss: 0.5765 - val_accuracy: 0.8756
Epoch 10/10
547/547 [==============================] - 22s 40ms/step - loss: 0.3992 - accuracy: 0.9348 - val_loss: 0.6057 - val_accuracy: 0.8727
Time taken to train the model:  480.6406178474426
```

Figure 19: Output log

Experiment and Obtained Result - LSTM + Pre-trained Embedding

- Plotting model loss and accuracy

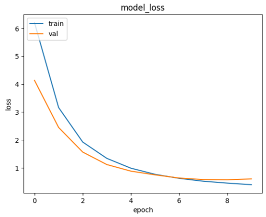
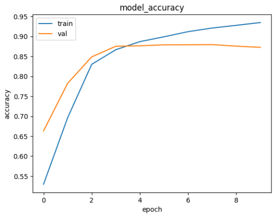


Figure 20: Accuracy

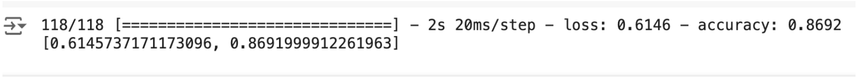


Experiment and Obtained Result - LSTM + Pre-trained Embedding

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)



```
118/118 [==============================] - 2s 20ms/step - loss: 0.6146 - accuracy: 0.8692
[0.6145737171173096, 0.8691999912261963]
```

Figure 22: 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.