Sentiment Analysis On IMDB Reviews Leveraging Transfer Learning And Neural Network Models Via Keras API

By

2024 | Armita Kharmandar

A Project Report Submitted In Partial Fulfillment Of The Requirements For The Degree Of

MASTER OF ENGINEERING
In the Department of Electrical and Computer Engineering

University of Victoria

2024 | Armita Kharmandar

Sentiment Analysis On IMDB Reviews Leveraging Transfer Learning And Neural Network Models Via Keras API

By

2024 | Armita Kharmandar

Supervisory Committee

Dr. Amirali Baniasadi

(Department of Electrical and Computer Engineering)

Dr. Alex Thomo

(the Department of Computer Science)

Table of Contents

1 Acronyms

NLP: Natural Language Processing

SLM: Statistical Language Models

NLMs: Neural Language Model

NNLM: Neural Network Language Model

OOV: Out-of-Vocabulary Handling

GPT: Generative Pre-trained Transformer

USE: Universal Sentence Encoder

BERT: Bidirectional Encoder Representations from Transformers

TFDS: TensorFlow Dataset

Adam: Adaptive Moment Estimation

SGD: Stochastic Gradient Descent

RMSProp: Root Mean Squared Propagation

SVM: Support Vector Machine

2 Abstract

This project focuses on sentiment analysis using a pre-labeled IMDB dataset containing 50,000 movie reviews. Our approach leverages transfer learning and a pre-trained neural network from TensorFlow Hub, specifically the NNLM model, to streamline the preprocessing stage. The sentiment classification model is built using the Keras API with a Sequential architecture. To design the model, several attempts have been made to improve the performance and overall accuracy and the finalized model includes first layer of an embedding layer that utilizes the pre-trained NNLM embeddings, followed by a three-layer network consisting of dense layer, dropout layer and another dense layer for classification respectively.

To compile the model, Adam optimizer and binary cross-entropy loss are defined while achieving an accuracy of almost 88%. In addition to this deep learning model, traditional sentiment analysis algorithms such as logistic regression, random forest, and SVM were also trained for comparison. Each attempt's output is visualized and finally in conclusion part it is discussed which of the models output the desired performance considering two factors of efficiency and accuracy. The results highlight that by incorporating transfer learning and Keras API, the overall model complexity and computational cost were significantly reduced while maintaining competitive accuracy.

3 Introduction

3.1 Background

In today's world of Natural Language Processing (NLP), sentiment analysis has become an essential tool for understanding opinions and emotions in large collections of text. It's all about figuring out whether the sentiment behind the words is positive or negative. From analyzing customer reviews to gauging brand reputation, sentiment analysis has a wide range of practical uses.

In this project, I am exploring how deep learning, especially transfer learning, can make sentiment analysis more efficient and accurate. By using pre-trained models from TensorFlow Hub, I aim to simplify the process while boosting model performance. The dataset I am working with is a well-known one: 50,000 movie reviews from IMDB, already labeled for sentiment, making it a solid choice for this kind of analysis.

3.2 Objective

The main objective of this project is to build an efficient sentiment analysis model using Keras API and transfer learning. By integrating a pre-trained text embedding model with a three-layer neural network, this study seeks to achieve competitive accuracy while minimizing computational complexity through different model design. This approach facilitates the developers focusing more on model design instead of academic details and theoretical implementation. Additionally, the project aims to compare this deep learning approach against traditional sentiment analysis methods such as Logistic Regression, Random Forest, and Support Vector Machines (SVM) to evaluate their respective strengths and weaknesses in compare with deep learning approaches.

3.3 Scope

This report covers the key methodologies, models, and techniques used in building the sentiment analysis system. It includes a comprehensive review of language models, with a focus on the NNLM embedding model. Additionally, the report discusses the performance evaluation of the deep learning model and traditional machine learning techniques, highlighting areas for future enhancement. The study serves as a guide for leveraging pre-trained embeddings and deep learning frameworks like Keras and TensorFlow for NLP tasks, providing both technical insights and practical applications.

4 Literature Review

4.1 What is Sentiment Analysis?

Sentiment Analysis is the process of using natural language processing (NLP), machine learning, and text analysis to identify and extract subjective information from text data. The goal is to determine the sentiment expressed in the text, usually classifying it as positive or negative.

Key Points:

Objective: Assess the sentiment behind opinions, reviews, social media posts, etc.

Applications: Product reviews, customer feedback, brand monitoring, and more.

Techniques: Machine learning models, rule-based systems, or a combination of both.

Output: A sentiment label (e.g., positive/negative), or a sentiment score indicating the intensity of sentiment.

4.2 What is language Model?

Language models are foundational tools in Natural Language Processing (NLP) that help systems understand and generate human language. They predict the probability of a sequence of words and are used in tasks such as text generation, machine translation, sentiment analysis, and more. Over time, language models have evolved significantly, leading to advancements from simple statistical approaches to sophisticated deep learning-based models.

4.2.1 Categories of Language Models

Language models can be broadly categorized based on the techniques they use. in the following paragraphs each technique is introduced.

Statistical Language Models (SLMs): Early models, such as n-grams, rely purely on statistical methods to predict the next word based on previous words. While effective for simple tasks, these models struggle with capturing long-term dependencies and context due to their fixed-size window of context.

Neural Language Models (NLMs): With the rise of deep learning, neural networks began to be used for modeling language. Unlike statistical models, neural language models are capable of capturing complex patterns and context across longer sequences. They use

embeddings to represent words as dense vectors, which can capture semantic meanings and relationships between words more effectively than traditional methods.

Transformers and Attention-Based Models: The development of transformer models, such as GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), and others, brought significant breakthroughs in NLP. These models leverage attention mechanisms, allowing them to focus on relevant parts of the input sequence while processing long contexts efficiently.

More specifically, in this project NLMs are utilized for sentiment analysis objective. NLMs marked a significant shift in how language is modeled. Unlike traditional statistical models, NLMs learn to represent words as vectors (embeddings) within a continuous space. This vectorization process allows the model to capture both semantic and syntactic relationships between words. These models learn to predict the likelihood of a word given its context by adjusting weights through backpropagation, allowing them to capture the patterns in text data.

4.2.1.1 Key Characteristics of Neural Language Models (NLMs):

Contextual Understanding: NLMs can process longer sequences and understand the context beyond fixed n-grams.

Embedding Representations: Words are represented as dense vectors, capturing their relationships in a continuous space.

Scalability and Flexibility: NLMs can be adapted and scaled for a wide range of NLP tasks.

4.2.2 NNLM: The Neural Network Language Model

Among the various NLMs, the **Neural Network Language Model (NNLM)** stands out as a foundational approach for pre-trained text embedding. The NNLM is particularly effective for tasks requiring contextual understanding and has been widely used for generating embeddings that can be fed into downstream models for tasks like sentiment analysis, classification, and more.

In this project NNLMs are widely used in preprocessing step. In the following paragraphs I will introduce the dataset and usage of transfer learning.

5 Methodology

5.1 Dataset:

In this project, sentiment analysis is applied to the IMDB dataset, which consists of 50,000 movie reviews, evenly split between positive and negative sentiments. The dataset is pre-labeled, with 25,000 reviews for training and 25,000 for testing with size of 80.23 MiB. Each data sample in the IMDB dataset includes a movie review and a label for that movie review. Labels consist of 0 or 1's. 0 indicates the negative comment and 1 indicates the positive one.

The IMDB dataset is a common benchmark for sentiment analysis tasks, making it a great choice for evaluating model's performance.

5.2 Data Collection:

TensorFlow Datasets (TFDS):

TFDS is a collection of ready-to-use datasets for machine learning and research. It provides a standardized interface for accessing and loading datasets in various formats like images, text, and audio.

5.3 Exploring the Dataset:

Data is split into 60% training and 40% validation which is 15000 samples for training, 10000 samples for validation and 25000 samples for testing. Figure 1 demonstrates the 3 first sample of dataset with their label of 0 or 1 indicating negative and positive respectively.

	Sample Text	Label
0	This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their worst role in history. Even their great acting could not redeem this movie's ridiculous storyline. This movie is an early nineties US propaganda piece. The most pathetic scenes were those when the Columbian rebels were making their cases for revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love affair with Walken was nothing but a pathetic emotional plug in a movie that was devoid of any real meaning. I am disappointed that there are movies like this, ruining actor's like Christopher Walken's good name. I could barely sit through it.	0
1	I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, being warm and comfortable on the sette and having just eaten a lot. However on this occasion I fell asleep because the film was rubbish. The plot development was constant. Constantly slow and boring. Things seemed to happen, but with no explanation of what was causing them or why. I admit, I may have missed part of the film, but i watched the majority of it and everything just seemed to happen of its own accord without any real concern for anything else. I cant recommend this film at all.	0
2	Mann photographs the Alberta Rocky Mountains in a superb fashion, and Jimmy Stewart and Walter Brennan give enjoyable performances as they always seem to do. But come on Hollywood - a Mountie telling the people of Dawson City, Yukon to elect themselves a marshal (yes a marshal!) and to enforce the law themselves, then gunfighters battling it out on the streets for control of the town? Nothing even remotely resembling that happened on the Canadian side of the border during the Klondike gold rush. Mr. Mann and company appear to have mistaken Dawson City for Deadwood, the Canadian North for the American Wild West. Canadian viewers be prepared for a Reefer Madness type of enjoyable howl with this ludicrous plot, or, to shake your head in disgust.	0

Figure 1- Data Exploration

5.4 Data preprocessing:

To perform sentiment analysis on this movie review dataset, we applied several preprocessing steps to clean and prepare the text data for modeling.

1. **Text Lowercasing** to convert all text to lower case to reduce the complexity caused by case differences.
2. **Removing Punctuation and Special Characters** that do not contribute to sentiment analysis.
3. **Tokenization** to split text into individual words(tokens)
4. **Embedding or Converting text to vector** for numerical representation of text so that model can process the text data.

To streamline the last process, embedding, NNLM as a pre-trained text embedding model is used while training a model from scratch is challenging and time consuming. The benefit of transfer learning in this process lies in its ability to allow developers and researchers to focus more on improving model accuracy and model design. By leveraging pre-trained models, much of the foundational learning is already accomplished, enabling efforts to be directed toward fine-tuning and optimizing the model for the specific task at hand.

Libraries like TensorFlow and PyTorch provide easy access to pre-trained models, allowing us to quickly apply them to our own data. Specifically in this project, I took advantage of TensorFlow library

In the following sections, I'll dive deeper into the model and how utilizing transfer learning via NNLM helped the model's efficiency.

5.5 Building the Model

The full model for training includes a two-layer neural network. The first layer is a pre-trained embedding model from TensorFlow Hub that outputs a 50-dimensional vector for tokenized words, which is then fed into the second Dense layer with 16 units. Finally, the model is compiled. In addition to this deep learning, other popular models for sentiment analysis including Logistic regression, Random Forest and support vector machine(SVM) are implemented and their performance is discussed.

5.5.1 TensorFlow Hub

TensorFlow Hub is a library and platform designed to share reusable machine learning models and components. While TensorFlow itself is a comprehensive framework for building

and training machine learning models from scratch, TensorFlow Hub focuses specifically on offering pre-trained models that can be easily integrated into new applications. These models, available as modules, include text embeddings, image classifiers, and more, making it easier to apply transfer learning for a wide range of tasks. Compared to TensorFlow, which requires building models layer by layer, TensorFlow Hub provides ready-to-use building blocks, allowing developers to leverage advanced models without extensive training processes. This makes TensorFlow Hub especially useful for tasks where rapid deployment and fine-tuning of models are more critical than designing architectures from the ground up.

Key Features:

- **Pre-trained Models**: Access models for tasks like image recognition, text embeddings, object detection, etc., without needing to train from scratch.
- **Easy Integration**: Models can be imported and used with a few lines of code using TensorFlow's `hub.KerasLayer()`.

In summary, TensorFlow Hub accelerates model development by providing reusable components that save time and resources, making it easier to experiment and deploy state-of-the-art models.

5.5.2 NNLM (Neural Network Language Model) Pre-trained Text Embedding Model

NNLM works by processing input text in several layers:

1. **Input Representation:** The input is a batch of sentences represented as a 1-D tensor of strings.
2. **Preprocessing:** The model preprocesses the input by splitting sentences on spaces to tokenize words.
3. **Out-of-Vocabulary (OOV) Handling:** Instead of discarding rare or unseen words or giving them generic 'unknown' representation, they are mapped to a fixed-size output through a hashing function. These outputs determine the similar words belong to a bucket. Therefore, model will generalize better, and it reduces the number of embeddings the model needs to learn.

   The embeddings for these hash buckets are not random and they are initialized using vectors from the remaining vocabulary. As a result, the model has a reasonable starting point to handle unseen data.

4. **Sentence Embedding:** The tokenized words are converted into embeddings. Next, they are combined using methods like `sqrtn` to generate a sentence-level embedding.
5. **Feed into a Neural Network:** The sentence embeddings are passed through two hidden layers, where the model learns patterns and associations relevant to the downstream task.

This model has different variation that can output each sample to 50 dimension or 128 dimensions. In addition, it is possible to normalize the data as well. It is provided in seven languages including English, Chinese, German, Indonesian, Japanese, Korean and Spanish.

Variation Selection is based on a trade-off between performance and computational efficiency. For this project, the en-dim50 variant is used while training the model on other variations is analyzed. In Figure 2, a Keras layer from TensorFlow Hub is utilized and the embeddings are demonstrated. The output for the first three samples from the training dataset showcasing how the embedding layer transforms the text into vectors.

```
hub_layer = hub. KerasLayer("https://tfhub.dev/google/nnlm-en-dim50/2", input_shape=[], dtype=tf.string, trainable=True)
hub_layer(train_examples_batch[:3])
```

```
<tf.Tensor: shape=(3, 50), dtype=float32, numpy=
array([[ 0.5423195 , -0.0119017 ,  0.06337538,  0.06862972, -0.16776837,
        -0.10581174,  0.16865303, -0.04998824, -0.31148055,  0.07910346,
         0.15442263,  0.01488662,  0.03930153,  0.19772711, -0.12215476,
        -0.04120981, -0.2704109 , -0.21922152,  0.26517662, -0.80739075,
         0.25833532, -0.3100421 ,  0.28683215,  0.1943387 , -0.29036492,
         0.03862849, -0.7844411 , -0.0479324 ,  0.4110299 , -0.36388892,
        -0.58034706,  0.30269456,  0.3630897 , -0.15227164, -0.44391504,
         0.19462997,  0.19528408,  0.05666234,  0.2890704 , -0.28468323,
        -0.00531206,  0.0571938 , -0.3201318 , -0.04418665, -0.08550783,
        -0.55847436, -0.23336391, -0.20782952, -0.03543064, -0.17533456],
       [ 0.56338924, -0.12339553, -0.10862679,  0.7753425 , -0.07667089,
        -0.15752277,  0.01872335, -0.08169781, -0.3521876 ,  0.4637341 ,
        -0.08492756,  0.07166859, -0.00670817,  0.12686075, -0.19326553,
        -0.52626437, -0.3295823 ,  0.14394785,  0.09043556, -0.5417555 ,
         0.02468163, -0.15456742,  0.68333143,  0.09068331, -0.45327246,
         0.23180096, -0.8615696 ,  0.34480393,  0.12838456, -0.58759046,
        -0.4071231 ,  0.23061076,  0.48426893, -0.27128142, -0.5380916 ,
         0.47016326,  0.22572741, -0.00830663,  0.2846242 , -0.304985  ,
         0.04400365,  0.25025874,  0.14867121,  0.40717036, -0.15422426,
        -0.06878027, -0.40825695, -0.3149215 ,  0.09283665, -0.20183425],
       [ 0.7456154 ,  0.21256861,  0.14400336,  0.5233862 ,  0.11032254,
         0.00902788, -0.3667802 , -0.08938274, -0.24165542,  0.33384594,
        -0.11194605, -0.01460047, -0.0071645 ,  0.19562712,  0.00685216,
        -0.24886718, -0.42796347,  0.18620004, -0.05241098, -0.66462487,
         0.13449019, -0.22205497,  0.08633006,  0.43685386,  0.2972681 ,
         0.36140734, -0.7196889 ,  0.05291241, -0.14316116, -0.1573394 ,
        -0.15056328, -0.05988009, -0.08178931, -0.15569411, -0.09303783,
        -0.18971172,  0.07620788, -0.02541647, -0.27134508, -0.3392682 ,
        -0.10296468, -0.27275252, -0.34078008,  0.20083304, -0.26644835,
         0.00655449, -0.05141488, -0.04261917, -0.45413622,  0.20023568]],
      dtype=float32)>
```

Figure 2 - Word Embedding Leveraging Transfer Learning

The model setting is set to en-dim50 variation mean each sample is first tokenized by splitting on space and removing punctuations, then they are converted into 50-dimensional vector. In this level the unseen and rare words will be hashed in some buckets and their embedding is initialized with an existing embedding in the model. After that, all the word embeddings are combined to sentence embedding.

Increasing the dimensionality will cause more rich detail and relationship which retain the sentiment of sentence but cause more computational costs. Variation selection is based on the priorities identified in the project and in this project the en-dim50 satisfy the requirements.

Next, these vectors are fed to second layer of the model. For this purpose, there is an API solution called Keras API which will increase the efficiency and simplicity significantly while it reduces the technical complexity in coding.

5.5.3 Keras API

The Keras API is a high-level, user-friendly interface for building and training deep learning models, integrated within TensorFlow. It provides simple functions and classes that allow developers to define neural networks without needing to worry about low-level implementation details.

On top of that, Keras provides pre-built layers, optimizers, and activation functions, allowing developers and researchers to concentrate on model design without getting bogged down by technical details. In this project, by utilizing the Keras API, we created a model where Keras layers are stacked via sequential function.

Figure 3 illustrates the model architecture, including **Sequential Function** which will group a linear stack of two layers into the model. Firstly, the embedding keraslayer then the dense layer. Next, the model is **Compiled**. Here loss functions and optimizers are specified and finally, the model is **Trained**. In summary, Keras API abstracts complex operations, while leveraging TensorFlow's powerful backend to handle the underlying computations.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 50)                48190600

 dense (Dense)               (None, 16)                816

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 48191433 (183.84 MB)
Trainable params: 48191433 (183.84 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 3- Model Architecture

5.6 Optimizing Model Development with Keras API and Transfer Learning

In general, within this two-layer deep learning model, designed using the Keras API and incorporating transfer learning, the Sequential API is used to build the architecture. The model starts with a text embedding layer, followed by a Dense layer, which is a fully connected layer. There, each neuron receives input from all neurons in the preceding layer. This Dense layer performs a weighted sum of inputs and applies the ReLU activation function. Finally, a one-dimensional Dense layer with a sigmoid activation function serves as the output layer, enabling the model to learn complex patterns effectively. This is followed by calling `*model.summary()*` to display the architecture, which includes the layer types, output shapes, and the number of parameters.

5.7 Trainable Parameters

The first layer, a pre-trained embedding layer (keras_layer), outputs a 50-dimensional vector representation and includes approximately 48 million parameters (48,190,600). The next layer is a fully connected Dense layer with 16 units, which adds 816 additional parameters. Finally, the output layer is another Dense layer with a single unit and sigmoid activation, responsible for producing the final classification output, and contributes 17 parameters.

In total, the model has around 48.2 million parameters, all of which are trainable. Given the extensive number of parameters, the model is capable of capturing complex patterns while remaining efficient enough for practical use.

With over 48 million parameters, we harness the power of transfer learning in the embedding layer to efficiently handle text data without needing to train from scratch. The pre-trained embedding layer significantly reduces the computational burden and speeds up model training by reusing learned representations. Additionally, the Keras API simplifies the integration of these complex layers, making the model both powerful and efficient. This combination allows us to quickly fine-tune the model while effectively capturing the intricacies of the data.

Key Points:

- **Sequentional API**: Groups a linear stack of two layers into the model.
- **Embedding Layer**: Contains many pre-trained parameters that speed up the preprocessing and learning process.
- **Dense Layers:** The trainable parameters in the Dense layers (833 total) are fully recognized by the model.
- **Model Summary**: Lists layer types, output shapes, and parameter counts, showing how each part of the model contributes to the overall architecture.

5.8 Model Challenge

One of the main challenges in implementing this model was managing the interaction between TensorFlow, TensorFlow Hub, and Keras. Although they are designed to integrate smoothly for simplicity, version compatibility issues can arise. The Sequential API, for instance, only accepts instances of keras.layer. Despite using compatible versions of TensorFlow and Keras, the model was unable to recognize the embedding layer as a valid keras.layer. This issue was resolved with a newer API called TF-Keras, a deep learning API written in Python, released on July 15th, 2024, which is fully compatible with TensorFlow 2.17.

6 Implementation

6.1 Compiling the Model

The model compilation process in TensorFlow/Keras involves defining three key components: the optimizer, the loss function, and evaluation metrics. In this model, I implemented the Adam optimizer, the binary cross-entropy loss, and accuracy as the evaluation metric.

1. **Optimizer: Adam**

   The Adam (Adaptive Moment Estimation) optimizer is a stochastic gradient descent (SGD) method which is widely used in deep learning. It combines the advantages of two popular methods: momentum, which adapts learning rates for each parameter, and RMSProp, which uses an exponentially decaying average of past squared gradients. It basically, adjust the learning rate adaptively for each parameter based on the gradients calculated for each parameter previously which cause faster convergence and more accurate than a fixed learning rate like SGD. Adam method is computationally efficient and is well suited for problem including large data and parameters. In the set of equations below, it is shown how the parameters are updated:

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

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

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- g_t: The gradient at time step t.
- m_t: The first moment estimate (exponentially weighted moving average of the gradient).
- v_t: The second moment estimate (exponentially weighted moving average of the squared gradient).
- β_1 and β_2: Hyperparameters controlling the decay rates for the moment estimates.
- α: The learning rate.
- ϵ: A small constant to avoid division by zero.

Adam works by maintaining two moving averages for each parameter: the first captures the mean of the gradients (first moment), and the second captures the uncentered variance (second moment). It then adjusts the learning rate for each parameter based on these values. Adam is particularly useful for sparse gradients and problems with noisy data,

making it a good choice for many deep learning tasks, including binary classification problems like this project.

2. **Loss Function: Binary Cross-Entropy**

   The binary cross-entropy loss function measures the difference between the predicted probability and the actual label for each sample through the equation below:

$$\text{Loss} = -\left[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})\right]$$

   Where:

   - y is the true label (either 0 or 1).
   - $\hat{y}$ is the predicted probability that the sample belongs to the positive class (1).

3. **Evaluation Metric: Accuracy**

   The evaluation metric chosen for the model is accuracy, which measures the proportion of correctly classified instances. In binary classification tasks, accuracy is often straightforward to interpret and implement. It compares the predicted label (rounded to 0 or 1) against the actual label, providing a simple but effective measure of how well the model is performing. While accuracy works well when classes are balanced, it may need to be complemented by other metrics, such as precision or recall, in cases with imbalanced datasets. In this model, accuracy is appropriate given that the binary classification problem is likely balanced, allowing the model to quickly assess the model's overall performance.

6.2 Training the Model

The model then is trained with the batch size 512, enabling the model to update its weights after each mini batch rather than after each individual sample. Training is through 10 epochs while during each epoch the model will be trained on the training dataset and evaluated the validation dataset. Following figures visualize the training process. Figure 5 demonstrates how the metrices changes over each epoch and figure 6 illustrates the trend of training and validation accuracy and loss.

```
30/30 [==============================] - 10s 250ms/step - loss: 0.6425 - accuracy: 0.6491 - val_loss: 0.5824 - val_accuracy: 0.7383
Epoch 2/10
30/30 [==============================] - 6s 191ms/step - loss: 0.5019 - accuracy: 0.7931 - val_loss: 0.4556 - val_accuracy: 0.8088
Epoch 3/10
30/30 [==============================] - 6s 196ms/step - loss: 0.3582 - accuracy: 0.8693 - val_loss: 0.3719 - val_accuracy: 0.8469
Epoch 4/10
30/30 [==============================] - 6s 192ms/step - loss: 0.2580 - accuracy: 0.9123 - val_loss: 0.3324 - val_accuracy: 0.8608
Epoch 5/10
30/30 [==============================] - 6s 193ms/step - loss: 0.1876 - accuracy: 0.9439 - val_loss: 0.3143 - val_accuracy: 0.8684
Epoch 6/10
30/30 [==============================] - 6s 195ms/step - loss: 0.1358 - accuracy: 0.9641 - val_loss: 0.3097 - val_accuracy: 0.8701
Epoch 7/10
30/30 [==============================] - 6s 192ms/step - loss: 0.0977 - accuracy: 0.9785 - val_loss: 0.3133 - val_accuracy: 0.8704
Epoch 8/10
30/30 [==============================] - 6s 191ms/step - loss: 0.0693 - accuracy: 0.9881 - val_loss: 0.3213 - val_accuracy: 0.8687
Epoch 9/10
30/30 [==============================] - 6s 190ms/step - loss: 0.0497 - accuracy: 0.9933 - val_loss: 0.3320 - val_accuracy: 0.8696
Epoch 10/10
30/30 [==============================] - 6s 191ms/step - loss: 0.0352 - accuracy: 0.9970 - val_loss: 0.3471 - val_accuracy: 0.8678
```

Figure 4- Training Process

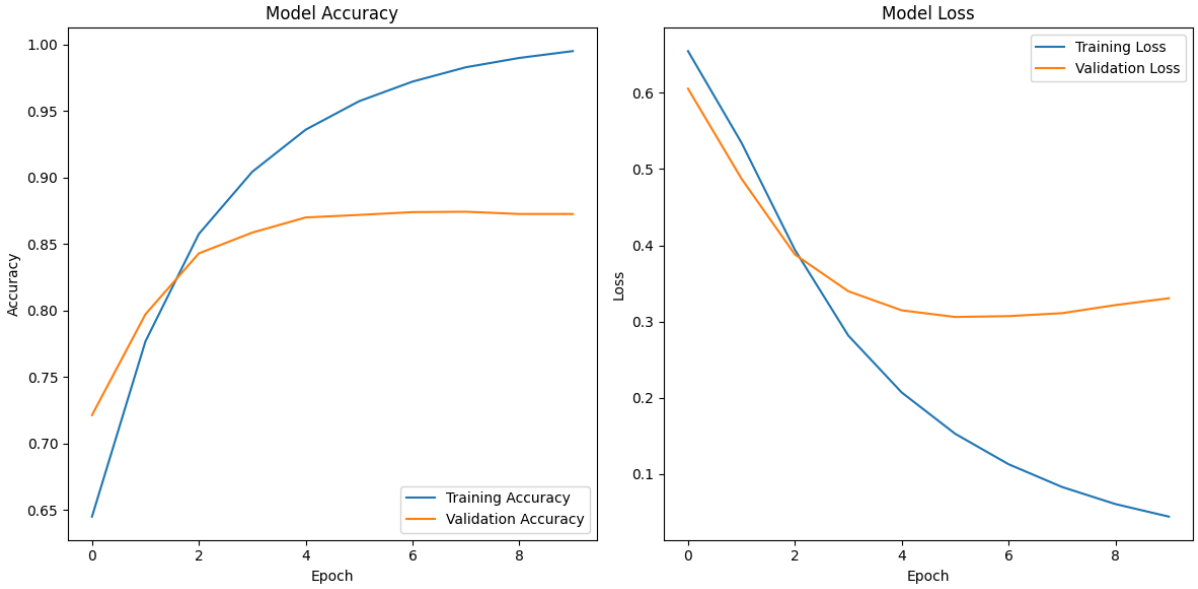

Figure 5- Training and Validation Trend

6.3 Analysis

The training and validation trends highlight important aspects of the model's performance over time. The **training accuracy** steadily increases, nearing almost 100%, while the **training loss** consistently decreases, showing that the model is effectively learning from the training data. However, the **validation accuracy** plateaus around 87% after the initial improvement, and the **validation loss** starts to increase after epoch 6. This suggests that while the model performs very well on the training data, it struggles to generalize to new data, a sign of **overfitting**. The model begins to memorize the training data instead of learning patterns that would apply to unseen examples. This overfitting is evident in the widening gap between training and validation performance. To improve generalization, techniques like early stopping, regularization, or data augmentation could be considered as effective solutions.

Through the next section, different attempts have been made to improve the model improvement.

7 Model Improvement

7.1 Second attempt

In my second attempt, I aimed to improve the model's performance by focusing on a key issue: its difficulty in handling negation, which was clear from the misclassified reviews shown in Figure 6. The model often struggled with sentences containing words like "not," "no," or "never," leading to incorrect sentiment predictions. To address this, I customized the data cleaning process by adding a "NOT_" prefix to words following negation terms, helping the model better capture the intended meaning. I also enhanced the model's design by adding a dropout layer, which I fine-tuned to a rate of 0.598. As shown in Figure 7, this adjustment significantly reduced overfitting. It is obvious that cross validation loss is happening mostly during the later training epochs which may be solved with some early stopping solution. In addition, without slowing down the training or greatly increasing the number of trainable parameters model demonstrated sufficient improvement. Early stopping could be one of the future steps to further refine the model, but this approach already shows a promising improvement in its ability to generalize effectively.

```
--------------------------------------------------
Text: As long as you keep in mind that the production of this movie was a copyright ploy, and not intended as a se
rious release, it is actually surprising how not absolutely horrible it is. I even liked the theme music.<br /><br
/>And if ever a flick cried out for a treatment by Joel (or Mike) and the MST3K Bots, this is it! Watch this with
a bunch of smart-ass wise-crackers, and you're in for a good time. Have a brew, butter up some large pretzels, and
enjoy.<br /><br />Of course, obtaining a copy requires buying a bootleg or downloading it as shareware, but if yo
u're here on the IMDb, then you're most likely savvy enough to do so. Good luck.<br /><br />And look for my favori
te part....where Dr. Doom informs the FF that they have 12 hours to comply with his wishes....and he actually gest
ures the number "12" with his finger while doing so....it's like "Evil Sesame Street"....hoo boy.<br /><br />...an
d of course Mrs. Storm declaring "Just look at you....the Fanstastic Four" is just so heartwarming....you'll laug
h, you'll cry.....<br /><br />So if you love schlocky Sci-Fi, this one's Fantastic For you!
True Label: 0
Predicted Label: 1
--------------------------------------------------
```

Figure 6- Model Misclassification Example

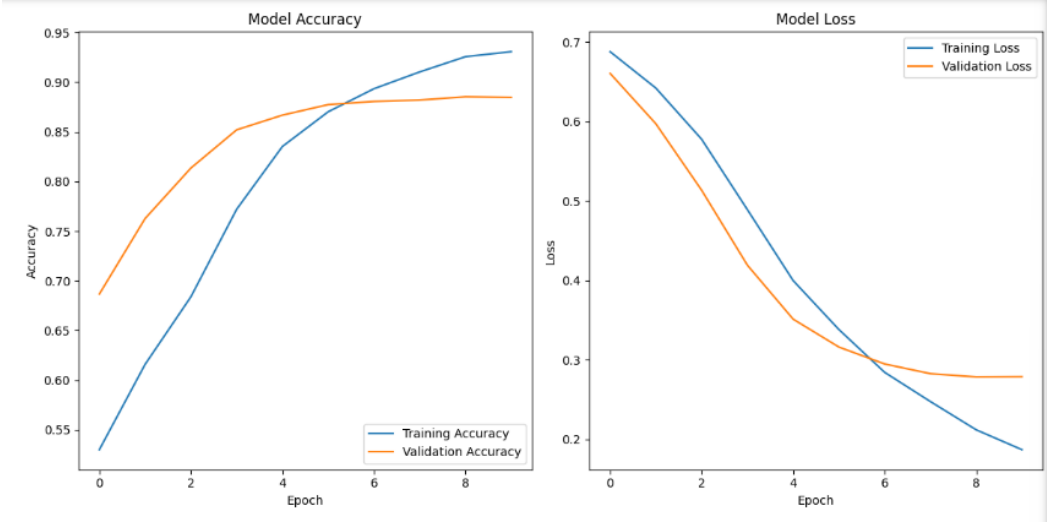

Figure 7- Training and Validation Trend of Improved Model

7.2 Next Attempt

In the following attempt, I explored a different variation of word embeddings to enhance the model's performance. While the initial approach utilized the NNLM embedding with a size of 50 and without normalization, this time I opted for a larger embedding size of 128 with normalization. The goal was to capture richer contextual details and improve the robustness of the model. However, as shown in Figure 8, while the issue of overfitting was mitigated, the trade-off became apparent: training times were significantly longer, and the number of trainable parameters increased substantially. Additionally, the validation loss began rising from epoch 3, and the validation accuracy did not match the levels seen in the previous model design. This suggests that despite the more sophisticated embedding, the model struggled to generalize effectively, highlighting a need to balance model complexity with performance.

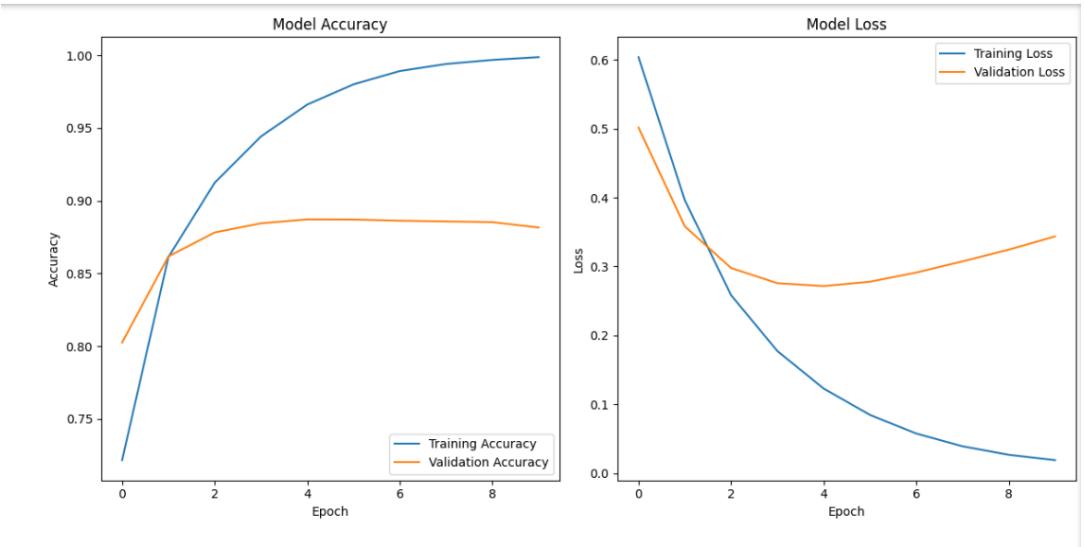

Figure 8 - Training and Validation Trend of Model with Larger Embedding Size with Normalization

7.3 Final Attempt

In the final attempt, I incorporated a transformer-based architecture for the word embedding model, specifically the Universal Sentence Encoder (USE). The USE is designed to capture sentence-level semantics by generating 512-dimensional embeddings for entire sentences, making it more robust compared to word-level embeddings used in previous models. This approach allows the model to better understand the overall context of a sentence rather than just individual words. Unlike the previous attempts where training spanned 10 epochs, this model was trained for only 3 epochs due to the significantly increased computational demand and a much larger number of trainable parameters which is demonstrated in figure 9. Despite these adjustments, the validation loss began to rise as early as the second epoch, peaking at a higher value of 0.3452, compared to 0.3133 in the initial model design. Although this model captured more complex patterns, it came with notable challenges, such as longer training times (figure 10) and increased resource requirements. Ultimately, despite these efforts, the final accuracy showing in figure 11 was not outstanding, highlighting the complexities and trade-offs involved in enhancing model performance with more advanced architectures.

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer_6 (KerasLayer)  (None, 512)               256797824

 dense_10 (Dense)            (None, 16)                8208

 dense_11 (Dense)            (None, 1)                 17

=================================================================
Total params: 256806049 (979.64 MB)
Trainable params: 256806049 (979.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 9 - Trainable Parameters of USE Model

```
Epoch 1/3
30/30 [==============================] - 97s 3s/step - loss: 0.0142 - accuracy: 0.9983 - val_loss: 0.3101 - val_accuracy: 0.8948
Epoch 2/3
30/30 [==============================] - 119s 4s/step - loss: 0.0078 - accuracy: 0.9995 - val_loss: 0.3301 - val_accuracy: 0.8945
Epoch 3/3
30/30 [==============================] - 214s 7s/step - loss: 0.0056 - accuracy: 0.9997 - val_loss: 0.3452 - val_accuracy: 0.8947
```

Figure 10- Training Epochs for USE Model

```
49/49 - 29s - loss: 0.3735 - accuracy: 0.8840 - 29s/epoch - 584ms/step
loss: 0.374
accuracy: 0.884
```

Figure 11 - USE Model Results

8 Evaluation

Figures below illustrate the models' performance during the evaluation step. Accuracy of the deep learning model is 0.857 which can be interpreted as a suitable output and may increase after the network refinement with regularization layers or early stopping. In addition, the results of implementing different popular algorithms for sentiment analysis like Logistic Regression, Random Forest and Support Vector Machine (SVM) are illustrated and compared in figures 8, 9 and 10 respectively.

```
49/49 - 28s - loss: 0.3620 - accuracy: 0.8567 - 28s/epoch - 562ms/step
loss: 0.362
accuracy: 0.857
```

Figure 12- Deep Learning Model Evaluation

```
Test Accuracy: 0.876

Classification Report:
              precision    recall  f1-score

           0       0.88      0.87      0.88
           1       0.87      0.88      0.88

    accuracy                           0.88
   macro avg       0.88      0.88      0.88
weighted avg       0.88      0.88      0.88
```

Figure 13- Logistic Regression Evaluation

```
Test Accuracy: 0.833

Classification Report:
              precision    recall  f1-score

           0       0.82      0.85      0.84
           1       0.84      0.82      0.83

    accuracy                           0.83
   macro avg       0.83      0.83      0.83
weighted avg       0.83      0.83      0.83
```

Figure 14- Random Forest Evaluation

```
Test Accuracy: 0.873

Classification Report:
              precision    recall  f1-score

           0       0.87      0.87      0.87
           1       0.87      0.87      0.87

    accuracy                           0.87
   macro avg       0.87      0.87      0.87
weighted avg       0.87      0.87      0.87
```

Figure 15- SVM Evaluation

8.1 Model Comparison

Here for better understanding, the outputs are compared, and their advantages and challenges are discussed:

1- **Deep Learning Model (Keras API + NNLM Embeddings):**

- **Advantages:**

- The use of pre-trained embeddings (NNLM) simplified feature extraction and reduced the complexity of model training.
- The deep learning model delivered acceptable accuracy while minimizing the need for extensive feature engineering.

- **Challenges:**

- The model's accuracy is slightly lower compared to traditional machine learning methods like logistic regression.

2- Logistic Regression

- **Advantages:**

- Simple and interpretable.
- High accuracy and balanced precision-recall scores across sentiment classes.

- **Challenges:**

- Requires manual feature extraction and preprocessing, which can be time-consuming.

3- Random Forest

- **Advantages:**

- Robust against overfitting due to ensemble averaging.
- Handles non-linear relationships well.

- **Challenges:**

- Slightly lower accuracy and unbalanced precision-recall for different sentiment classes.

Support Vector Machine (SVM)

- **Advantages:**

- High accuracy with well-balanced precision and recall scores.

- **Challenges:**

- Computationally intensive and slow to train, particularly on large datasets.

In summary, while traditional algorithms like logistic regression and SVM offer strong performance with high accuracy, the deep learning model demonstrates that incorporating transfer learning in sentiment analysis can achieve competitive results with significantly less feature engineering. Given the trade-off between training time and performance, the deep learning model strikes a good balance between simplicity and accuracy, making it a suitable choice for scalable sentiment analysis tasks.

9 Future works

There are some plans aiming to both improve the model's performance and extend its application to real-world scenarios while leveraging the strengths of transfer learning and accessible deep learning frameworks.

1- **Exploring More Advanced Pre-Trained Models like BERT and GPT**, which have shown superior performance in various NLP tasks and hold the potential to further enhance accuracy in sentiment analysis.
2- **Incorporating More Complex Model Architectures such as bidirectional LSTMs or attention mechanisms** that can enhance the model's ability to capture context and sentiment more accurately, especially in cases where the sentiment is more complex.
3- Discover more about popular sentiment analysis APIs **like Google Cloud Natural Language and IBM Watson**

10 Conclusion

This project demonstrates the advantages of integrating transfer learning and the Keras API for sentiment analysis. Using pre-trained NNLM embeddings minimized the need for manual feature engineering, allowing for efficient model development with competitive results. Subsequent attempts to refine the model—such as handling negations, adjusting embedding sizes, and integrating the Universal Sentence Encoder—highlighted the trade-offs between performance, training time, and model complexity. While traditional methods like logistic regression and SVM remain strong contenders, the deep learning approach with transfer learning proved to streamline the process without sacrificing accuracy. The combination of pre-trained models and Keras offers a powerful, accessible, and scalable solution for sentiment analysis, striking a balance between performance, ease of implementation, and computational efficiency.

11 Bibliography

- A. Chinnalagu and A. K. Durairaj, "Comparative Analysis of BERT-base Transformers and Deep Learning Sentiment Prediction Models," 2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART), Moradabad, India, 2022, pp. 874-879, doi: 10.1109/SMART55829.2022.10047651. keywords: {Deep learning;Analytical models;Social networking (online);Bit error rate;Companies;Predictive models;Transformers;Pre-trained Transformers Models;BERT;FastText;Comparative Analysis;Deep Learning Sentiment Models}

- M. Trupthi, S. Pabboju and G. Narasimha, "Sentiment Analysis on Twitter Using Streaming API," 2017 IEEE 7th International Advance Computing Conference (IACC), Hyderabad, India, 2017, pp. 915-919, doi: 10.1109/IACC.2017.0186. keywords: {Training;Sentiment analysis;Twitter;Data mining;Matched filters;Information filters;Sentiment Analysis;Streaming API;Twitter},

- TensorFlow. (n.d.). *TensorFlow Hub: A library for reusable machine learning modules*. TensorFlow. Retrieved August 23, 2024, from https://www.tensorflow.org/hub

- Keras. (n.d.). *Getting started with Keras*. Keras. From https://keras.io/getting_started/

- Stack Overflow. (2020, July 1). *TensorFlow compatibility with Keras* [Discussion post]. Stack Overflow. https://stackoverflow.com/questions/62690377/tensorflow-compatibility-with-keras

- TensorFlow. (n.d.). *TensorFlow: An open-source machine learning framework for everyone*. TensorFlow. Retrieved August 23, 2024, from https://www.tensorflow.org/

- PyPI. (n.d.). *tf-keras* [Python package]. PyPI. Retrieved August 23, 2024, from https://pypi.org/project/tf-keras/

- Codex. (2021, May 20). *Sentiment analysis with TensorFlow Hub*. Medium. https://medium.com/codex/sentiment-analysis-with-tensorflow-hub-678c30ac79a2

- Arize AI. (n.d.). *Binary cross-entropy/log loss: A comprehensive guide*. Arize AI. Retrieved August 23, 2024, from https://arize.com/blog-course/binary-cross-entropy-log-loss/

- Kumar, K. (n.d.). *Getting to know Adam optimization: A comprehensive guide*. LinkedIn. Retrieved August 23, 2024, from https://www.linkedin.com/pulse/getting-know-adam-optimization-comprehensive-guide-kiran-kumar/

- slds-lmu. (2020). *Introduction to Deep Learning for NLP*. Retrieved from https://slds-lmu.github.io/seminar_nlp_ss20/introduction-deep-learning-for-nlp.html

- Keras Team. (n.d.). *Hashing layer.* Keras. Retrieved from https://keras.io/api/layers/preprocessing_layers/categorical/hashing/