DeepSentiment

A study on Deep Learning Models for Sentiment Classification

An Cao, Ryan Kim, Daniel Xiong

May 2024

1 Introduction

Sentiment classification is a crucial task in natural language processing, with applications in social media analysis and customer feedback processing. This project implements and compares three approaches: Graph Convolutional Networks (GCN), Multi-Channel Convolutional Neural Networks (MC-CNN), and Adaptive Multi-Channel (AM-GCN). The project implements an RNN to compare the performance of these 3 models which will allow us to effectively evaluate their performance and identify the best model for sentiment classification.

2 Data

For our data, we used the Stanford Large Movie Review Dataset [2]. This is a commonly used dataset for binary sentiment classification. It contains 50,000 highly polar movie reviews (labels are 1 or 0, 1 if positive 0 if negative). The dataset provides a substantial amount of data for sentiment analysis tasks, allowing for more robust model training. Our preprocessing includes removing rare words, removing stop words and punctuations, and tokenizing the text.

3 Methodology

3.1 Data Preprocessing

The dataset consisted of a movie review as an input and a label that was either 0 or 1. We tokenized the inputs with a predefined maximum vocabulary size (20,000), which served to convert the raw textual data into sequences of integers. The integers used to represent each of the words were constructed so that the most frequently used words had a smaller number, and the least frequently used words had a larger number. We did this in order to reduce memory usage and make the graph construction process easier.

Github link

Following tokenization, we padded the inputs to ensure consistency in input size. This involved setting a maximum sequence length ('MAX_LEN') and padding shorter sequences with zeros, thereby standardizing the input format across the dataset.

To get rid of noise, common stop words such as "and", "this", and "the" were identified and systematically removed as these words have little semantic meaning and does not signifcantly alter the logic of a sentence. As a result, we decided to exclude them from the dataset using the implementations provided by nltk to improve model performance. Similarly, we decided to get rid of rare words which would be relatively less significant and computationally expensive to include.

For training, we added shuffling and train-test splitting which were implemented to enhance model generalization and mitigate overfitting. Shuffling the dataset ensures that the model encounters diverse data during training, while the train-test split facilitated unbiased evaluation by separating a portion of the data for testing purposes.

3.2 Models

3.2.1 Recurrent Neural Networks (RNN)

RNNs are a class of neural networks designed to process sequential data by maintaining a memory of previous inputs. This architecture makes them particularly well-suited for tasks involving text data, where understanding the context and temporal dependencies between words is crucial. We used RNN as our baseline model to compare accuracy.

Model Architecture

Our RNN model was constructed using TensorFlow's Keras API. Within this architecture, we employed multiple layers, with each layer comprising Long Short-Term Memory (LSTM) cells. LSTMs are specialized units capable of capturing long-range dependencies in sequential data, making them ideal for sentiment analysis tasks. The key components include

1. Embedding Layer: The input text data was first transformed using an embedding layer, which mapped each word to a high-dimensional vector representation. This transformation facilitated the neural network's ability to process textual information in a vector space.
2. LSTM Layer: Following the embedding layer, the sequential data flowed through multiple LSTM layers. These layers were responsible for processing the input sequences while preserving information about the temporal dependencies between words. By retaining memory of previous inputs, the LSTM cells captured contextual information crucial for sentiment analysis.
3. Dense Layer with Sigmoid Activation: The final layer of the RNN model consists of a dense layer with a sigmoid activation function. This layer

produces the sentiment prediction for each input sequence, with the sigmoid function ensuring output values between 0 and 1, representing the likelihood of positive or negative sentiment.

3.2.2 Multi-Channel Convolutional Neural Network (MC-CNN)

A Multi-Channel Convolutional Neural Network (MC-CNN) [1] consists of multiple parallel channels, each processing the input text in a different way (e.g., using different filter sizes or kernel sizes). The outputs of these channels will be concatenated and fed into a fully connected layer for classification. This architecture is known for its effectiveness in capturing different aspects of the input data.

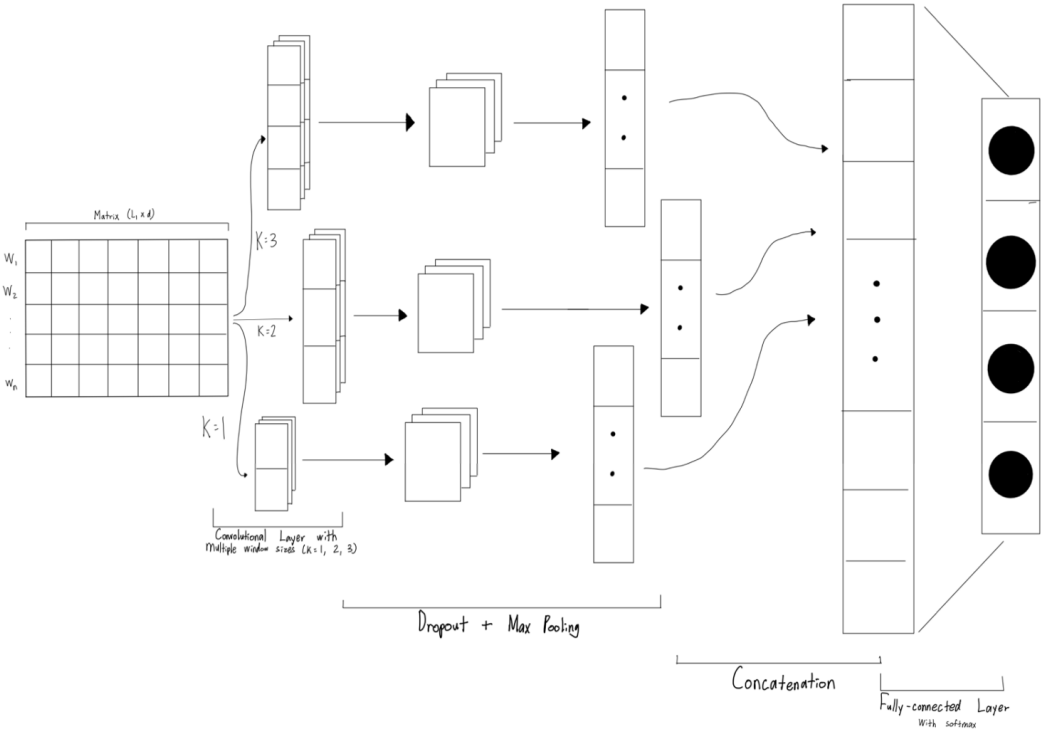

Figure 1: Overview of MC-CNN Architecture

Our model consists of three parallel channels. Channel 1 is a one-dimensional convolutional layer with a kernel size of 4. Channel 2 and 3 are also one-dimensional convolutional layers with a kernel size of 6 and 8, respectively. In each channel, the embedded input is passed through the convolutional layer, then followed by a dropout layer, max pooling layer, and a flattening layer. The outputs from all three channels are then concatenated and passed through two dense layers. The final dense layer has sigmoid activations, returning the final output being a probability score indicating the sentiment of the input text.

The MC-CNN model is designed to leverage multiple channels to extract different types of features from the input data, which can improve the accuracy of movie review emotion and sentiment recognition. The use of multiple channels with

different kernel sizes allows the model to capture both local and global features of the input data. The concatenation of the outputs from all three channels allows the model to combine the features extracted from each channel, which can improve the generalization ability of the model.

3.2.3 Graph Convolutional Networks (GCN)

GCNs[4] are neural networks designed to work with graph-structured data, which is particularly useful for tasks where the input can be represented as a graph. In our case, we will represent the text data as a graph where nodes represent words or tokens, and edges represent relationships between them (e.g., co-occurrence). The GCN will learn to perform sentiment classification based on this graph representation of the text.

To adapt GCNs for sentiment classification, we constructed a large and heterogeneous text graph in the form of an adjacency matrix, where each row/column represents a word. Each row of the matrix represents the frequency in which another word appeared in the same window as the word represented by the row. We then feed the constructed text graph into a two-layer GCN architecture, where message passing occurs among nodes up to two steps away. In the final layer, GCN model employs a softmax classifier to predict sentiment labels based on the learned node embeddings.

3.2.4 Adaptive Multi-Channel Graph Convolutional Networks (AM-GCN)

Adaptive Multichannel GCN [3] is an extension of the GCN model that extracts information from the graph through multi-channel graph convolution and uses an adaptive attention layer to learn weights for each channel. This allows the model to focus on the most relevant information for each node, improving node representation learning. Unlike conventional GCNs, which often smooth node features or impose low-pass filters during information propagation, AM-GCN acknowledges the complementary nature of feature and topology similarities.

AM-GCN achieves this by employing specific convolution modules tailored for propagating features over both topology and feature spaces. These modules extract distinct embeddings from each space, capturing both local and global information. Moreover, a common convolution module with parameter sharing is utilized to capture shared information between the two spaces, promoting consistency in the learned representations.

A key aspect of AM-GCN is its attention mechanism, which dynamically fuses the embeddings extracted from feature and topology spaces. This attention mechanism allows AM-GCN to adaptively extract the most correlated information for classification tasks, enhancing its discriminative power.

Additionally, AM-GCN integrates consistency and disparity constraints to ensure the coherence and independence of learned embeddings. These constraints play a crucial role in enhancing the robustness and interpretability of the learned representations.

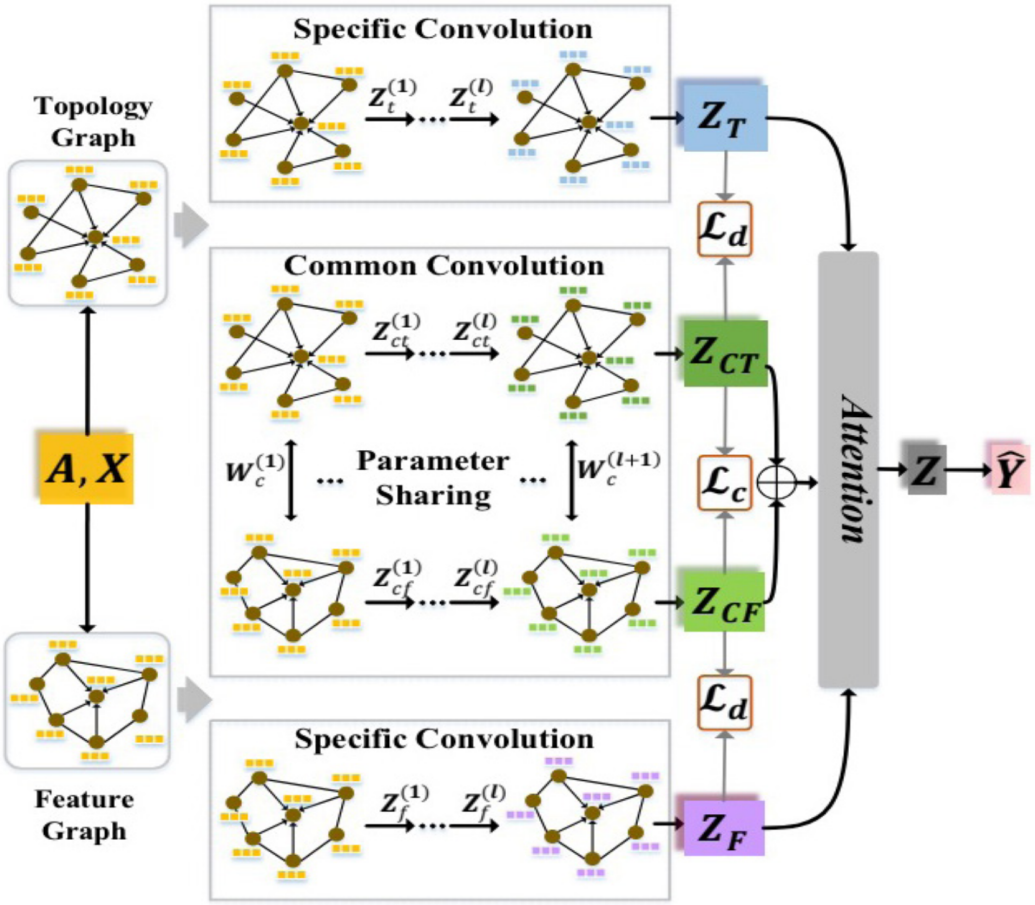

Figure 2: Overview of AMGCN Architecture

3.3 Training and Testing

- Training: We have trained all 3 models on the datasets proposed in their respective original papers. In particular, we trained our MC-CNN model on the STS-Gold dataset; GCN and AM-GCN on ACM and CITESEER datasets. We have trained RNN and MC-CNN on the Stanford Large Movie Review Dataset and are currently working on training GCN models on this dataset. We have finished building the graph to train the model on, but although our graph has attempted to minimize computational costs, the size of the dataset and the structure of the graph have posed challenges to run our model.
- Metrics: For assessing our model's performance, we will primarily look at accuracy, precision, recall, and F1-score.

4 Results & Discussion

We ran our baseline model RNN and model MC-CNN on the Stanford Large Movie Review Dataset (IMDB) and STS-Gold Dataset:

Datasets	RNN	MC-CNN
IMDB	81.5%	79.7%
STS-Gold	79.6%	85.7%

On the IMDB dataset, RNN slightly outperforms MC-CNN while MC-CNN outperforms RNN on the STS-Gold dataset. This sheds light on the datasets' text pattern. We speculate that the IMDB dataset contains long sentences and has more sequential relationship, which makes it more suitable for RNN. On the other hand STS-Gold dataset has shorter sentences, which works well for MC-CNN as it is better at capturing different features of the text.

We ran the graph convolutional models on the ACM and CITESEER datasets and compared with the original implementation performance:

Datasets	GCN	AM-GCN	AM-GCN (original)
ACM	84.4%	86.0%	90.9%
CITESEER	67.2%	69.3%	73.5%

On both ACM and CITESEER, our AM-GCN model works comparably to the original implementation, with a slight decrease in accuracy. We are working on tuning parameters to achieve equivalent results. Our GCN model has lower accuracy than AM-GCN as expected since AM-GCN has an attention module that helps with the model's ability to focus on relevant nodes and perform better predictions.

5 Challenges

In building a Graph Convolutional Network (GCN) model, we have run into several challenges. Firstly, constructing an appropriate graph structure is paramount. This entails defining the relationships between nodes, which can be complex and dynamic, particularly in real-world datasets. Ensuring the fidelity of this graph structure is crucial for the GCN's effectiveness in capturing meaningful information from the data.

Secondly, the challenge of handling large vocabulary sizes can significantly impact training times. With extensive vocabularies, the computational complexity of processing and updating embeddings increases substantially. Efficient strategies, such as subsampling or hierarchical approaches, may be necessary to mitigate this challenge and expedite training without sacrificing model performance.

Integrating different models into a cohesive framework poses another challenge. Ensuring the compatibility of different models to the processed data required careful planning and an understanding of how all of the models work.

Finally, tuning hyperparameters for optimal performance presents an ongoing challenge in machine learning model development. Selecting the right combination of hyperparameters can significantly impact the model's effectiveness and generalization ability. This was especially challenging due to the lack of time, and the

limitations of our machines, which made every run of the model take a significant amount of time.

6 Reflection

1. Project Outcome: Overall, our project met our expectations. We made significant progress and accomplished most of what we set out to do in terms of implementing the model and addressing technical challenges. While we didn't achieve all our stretch goals, we met our base and target goals to a satisfactory extent.

2. Expectations vs Reality: Since we didn't have concrete results to evaluate the GCN models' performance on the movie review dataset, it's difficult to say whether it worked out as expected. However, in terms of code functionality and implementation, we were able to overcome obstacles and refine our approach as needed.

3. Approach: Our approach evolved over time as we encountered various challenges and learned more about the intricacies of the problem domain. We made several pivots in our implementation strategy, particularly regarding data preprocessing and model architecture. For example, we noticed that given our resources, it was impossible to work with a larger vocabulary size due to RAM limitations, so we tried to reduce the vocabulary size and eliminate more rare words. If given the opportunity to redo the project, we would try to optimize the graph building part and test the models on a larger variety of datasets. We also decided not to use document nodes and only used word nodes due to RAM limitations, since our dataset was relatively large.

4. Areas for Improvement: Given more time, there are several areas we could further improve upon. These include fine-tuning hyperparameters, optimizing model performance, conducting comprehensive experiments to evaluate the model's effectiveness, and exploring alternative architectures or techniques. Additionally, enhancing the scalability and efficiency of our codebase could be beneficial for handling larger datasets.

5. Key Takeaways: This project provided valuable insights into the challenges and complexities involved in building machine learning models, particularly in the domain of natural language processing. We learned the importance of collaboration, communication, and adaptability when working on a team project. Additionally, debugging skills and familiarity with tools like GitHub and TensorFlow were strengthened through practical application.

References

[1] Jumayel Islam, Robert E. Mercer, and Lu Xiao. Multi-channel convolutional neural network for twitter emotion and sentiment recognition. *IEEE Transactions on Affective Computing*, 11(4):755–766, 2020.

[2] StanfordNLP. Imdb dataset. https://huggingface.co/datasets/stanfordnlp/imdb, 2024.

[3] Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. Am-gcn: Adaptive multi-channel graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11108–11115, 2020.

[4] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7370–7377, 2019.