CMPSC 448 — Final Project

Group Member: Quan Shi, Keyu Lu, Zehao Chen, Chunxu Wang

Task & dataset & preprocessing

Task

The task at hand is Sentiment Analysis on Movie Reviews, aimed at classifying the sentiment of sentences extracted from the Rotten Tomatoes dataset. The objective is to discern whether a given sentence expresses positive or negative sentiment.

Dataset

The dataset utilized for this project is the Movie Review data from Rotten Tomatoes, encompassing a collection of 10,662 phrases split equally between positive and negative sentiment. The dataset is structured and conveniently provided by Kaggle, offering pre-segmented sentiment-labeled phrases.

Data Split: As the dataset lacked predefined training/test splits, a 90/10 train-dev split was employed, with 10% of the data allocated for the development set to evaluate model performance.

Cross-Validation: To mitigate overfitting concerns due to the relatively small dataset, a 10-fold cross-validation strategy was implemented.

Data Structure: The dataset comprises tab-separated files containing phrases from the Rotten Tomatoes dataset, each phrase associated with a phrase ID and a sentence ID. Additionally, to prevent repetition, common phrases or words appear only once in the dataset.

Preprocessing

There are some steps for data preprocessing:

1. Load positive and negative sentences from raw data files.
2. Clean the text using predefined methods.
3. Pad each sentence to a uniform length of 59 words, using special <PAD> tokens for shorter sentences.
4. This uniformity allows efficient batching, as each batch requires sentences of the same length.
5. Construct a vocabulary index, mapping each word to an integer from 0 to 18,765 (the total number of unique words in the vocabulary).
6. Convert each sentence into a vector of integers based on this vocabulary mapping.

There are also some recommended methods like:

Shuffling: To ensure randomness and prevent inherent ordering biases, the sentence order was randomized.

Benchmarking: The train/test split was preserved for benchmarking purposes, although the sentence sequences were shuffled. This facilitates fair comparison and evaluation of model performance while avoiding bias introduced by specific sequence orders.

Implementation & Architecture

In this project, we choose two deep learning systems that we have learned in the class: Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN).

Convolutional Neural Network (CNN) Architecture:

Embedding Layer: Embedding words into low-dimensional vectors.

Convolutional Layers: Utilizing multiple filters of varying sizes (3, 4, 5) to perform convolutions. Each filter generates features.
Max-Pooling Layer: Transforming the output of the convolutional layers into a longer feature vector and applying dropout for regularization.
Softmax Layer: Employed for classification, assigning probabilities to each class.

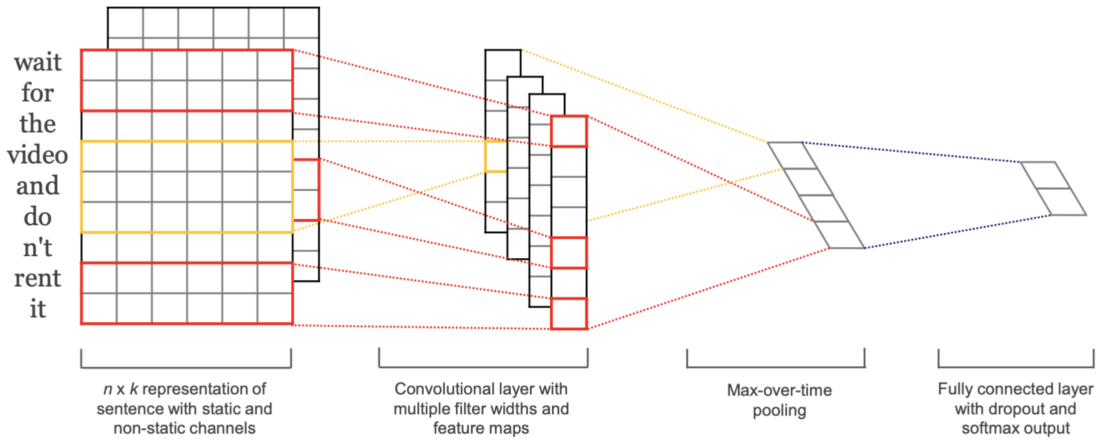
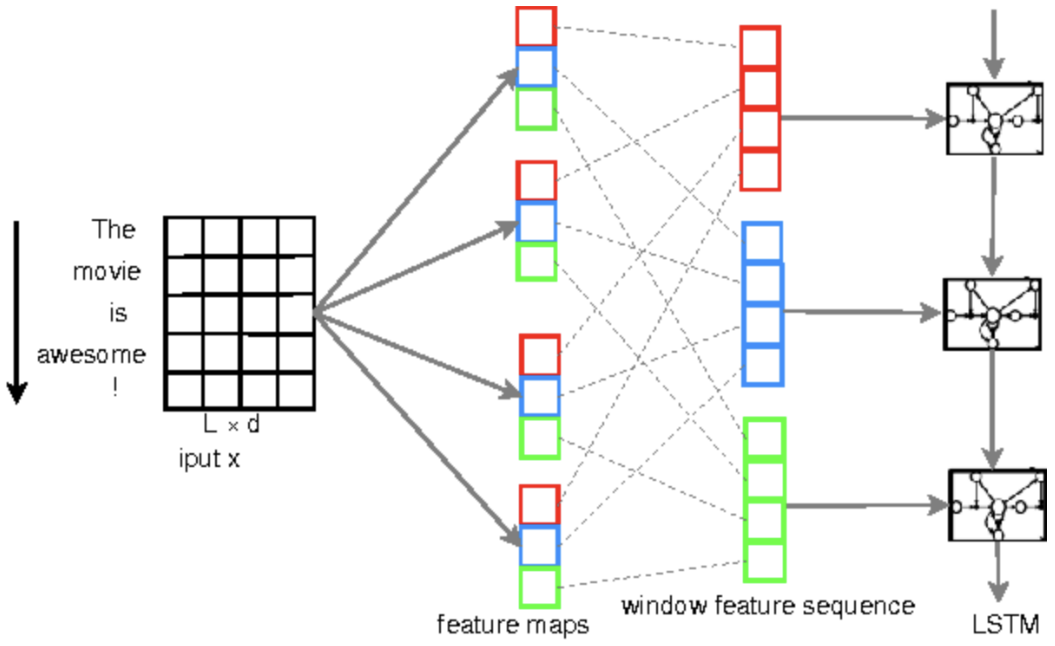

wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: Model architecture with two channels for an example sentence.

Recurrent Neural Network (RNN) Architecture:
Embedding Layer: Similar to CNN, converting words into embeddings.
RNN Layer: Processing the embedded words through the RNN. Typically, the output corresponding to the last word serves as the sentence's feature vector.
Fully Connected Layer + Softmax: Predicting probabilities for each label via a fully connected layer followed by softmax activation.
Loss Function and Optimization: Calculating loss using cross-entropy and updating parameters through optimization.



The primary difference between CNN and RNN lies in their approach to feature extraction from sentences. While CNN uses convolution and pooling for feature extraction, RNN employs sequential processing through recurrent connections.

Both architectures follow a similar flow: embedding the words, extracting features, and performing classification. However, the manner in which they extract features from sentences is distinct—the CNN extracts features using convolutions, while the RNN does so by processing the sequence of words through recurrent connections.

Training Detail

Convolutional Neural Network (CNN)：textCNN

File Structure:

Data Preparation: Loading and preprocessing the data, including loading text data from files, cleaning text strings, splitting into training and validation sets, and building the vocabulary.
Model Initialization: Initializing the CNN model with various parameters like sequence length, embedding size, filter sizes, number of filters, dropout probabilities, etc.
Training Loop: The actual training loop where batches of data are fed into the model.
For each batch:
Training Step: Performing a single training step by running a TensorFlow session and optimizing the model's parameters to minimize the defined loss function.
Evaluation Step: Periodically evaluating the model's performance on the validation set to monitor its accuracy and loss on unseen data.
Checkpointing: Saving the model's parameters periodically as checkpoints to restore or continue training later.
TensorBoard Logging: Logging training summaries such as loss, accuracy, and gradients to be visualized in TensorBoard for monitoring the training progress.
Model Saving: Saving the trained model to be used for inference or further training.

Recurrent Neural Network (RNN): textRNN
File Structure:
Data Loading and Preprocessing:
Loads data from positive and negative files.
Cleans the text data.
Splits the data into training and validation sets.
Builds a vocabulary based on the training data.
Model Architecture:
Uses an LSTM-based RNN for text classification.
Utilizes an embedding layer, followed by an LSTM layer.
Employs dropout for regularization.
Defines loss (softmax cross-entropy) and optimization (Adam) functions.
Sets up accuracy calculation.

Training Loop:
Initializes TensorFlow session and variables.
Runs training steps iteratively for each batch of data.
Evaluates the model periodically on the validation set.
Saves model checkpoints at certain intervals.
Logging and Summaries:
Logs training details like loss, accuracy, and steps.
Writes summaries for TensorBoard visualization.
Miscellaneous:
Includes some utility functions like writing results to a file.
Uses TensorFlow's tf.flags for setting up parameters.

The results & observations & conclusions

After training and running the code, we get the result: Loss of CNN is 0.51, accuracy of CNN is 0.72 and loss of RNN is 0.57, accuracy of RNN is 0.64.
So we can make some conclusion: On the Rotten Tomatoes dataset, the Text CNN model performs better relative to the Text RNN model.
The Text CNN model is slightly better than the Text RNN model in both accuracy and F1 Score.
The Text CNN model takes less time to train than the Text RNN model, improving efficiency while maintaining performance.
The reason why Text CNN may perform better on sentiment analysis tasks can be attributed to the following points:

Local feature extraction capability: CNN can capture local features more effectively in text data. The convolution kernel can slide in the text at different sizes to capture

features of different lengths. For sentiment analysis, some emotionally related words or phrases may appear in different positions of the sentence, and Text CNN is more likely to quickly capture these features to better understand the emotional tendency of the sentence.
Parallel computing capability: Compared with RNN, CNN has better parallel computing capabilities when processing text data. This makes CNN more efficient when processing long sequence data, because CNN can process different parts at the same time, while RNN needs to process each word step by step in sequence, making it difficult to achieve parallelization.
Insensitive to sequence length: In text classification tasks, CNN is less sensitive to changes in sequence length than RNN. For very long texts, RNN may encounter the problem of vanishing or exploding gradients, while CNN can cope with this situation better and maintain relatively stable performance.
Parameter sharing: CNN uses convolution kernels to extract features from text fragments at different locations. This parameter sharing method can reduce the amount of model parameters and improve model generalization capabilities and training efficiency.

The challenges & obstacles

- The first difficulty we encountered was spending a lot of time learning and understanding RNN and CNN basic concepts. In addition to the knowledge from the class, we also consulted a lot of online resources.

- Then, applying the abstract knowledge of RNN and CNN to write models in Python was also very challenging for us. Fortunately, we had help from reference articles.

- We all know that debugging is an inevitable part of programming. During our project development, debugging was omnipresent, and we spent a lot of time on it.

- Overfitting: RNNs can easily overfit to the training data, especially if the network is large and the training dataset is not sufficiently large or diverse. Our dataset is not very large, so there may overfitting problem in our result.

- Positional Invariance: While useful in image processing, CNN's characteristic of positional invariance can be a drawback in NLP since word order is crucial for understanding the meaning.

Reference

https://ieeexplore.ieee.org/document/9850878/references#references
https://dennybritz.com/posts/wildml/implementing-a-cnn-for-text-classification-in-tensorflow/
https://github.com/hthuwal/sign-language-gesture-recognition/blob/master/Final%20Report.pdf