Diego Toribio
Professor Sable
Project 2: Deep Learning Project
ECE - 467 Natural Language Processing

1. Introduction

For this project, I chose to work on text categorization using Corpus 1 from the datasets provided in Project #1, which consists of both a training set and a test set organized within the `TC_provided` directory. The dataset consists of text documents and their corresponding labels, specified in two files: `corpus1_train.labels` for the training data and `corpus1_test.labels` for the test data. Each line in these label files contains the relative path to a text file and its associated label.

To convert the data into the proper format for my system, I developed a `LoadDataset` class. This class reads the label files and processes each entry by constructing the full path to the text files based on the provided relative paths. It then reads the content of each text file and pairs it with its associated label. Specifically, the `load_dataset` method iterates over each line in the label files, extracts the file paths and labels, reads the text content, and aggregates the data into a list of dictionaries containing "`text`" and "`label`" keys. This list is then converted into pandas DataFrames for both the training and test sets.

The DataFrames are stored within the `LoadDataset` class and are readily accessible for further preprocessing by the `Processor` class. This design choice facilitated a smooth transition from raw data to a format suitable for tokenization, encoding, and model training within my system. By organizing the data in this way, I ensured that the text documents and labels were accurately and efficiently prepared for the machine learning pipeline.

2. Discussion

An experiment was conducted to explore different LSTM architectures and their impact on performance. Specifically, unidirectional, bidirectional, and stacked LSTM models (with 2 and 3 layers) were compared using both learned embeddings and pre-trained Word2Vec embeddings. For models with learned embeddings, the bidirectional LSTM achieved the highest test accuracy of **69.80%**, indicating that incorporating context from both past and future inputs enhances performance when embeddings are learned from scratch. In contrast, for models utilizing Word2Vec embeddings, the unidirectional LSTM attained the best test accuracy of **70.69%**, slightly outperforming its bidirectional counterpart. Overall, models using Word2Vec embeddings consistently outperformed those with learned embeddings, suggesting that using pre-trained embeddings captures more meaningful semantic information, thus improving classification accuracy.

To further optimize model performance, hyperparameters such as learning rates and dropout rates were tuned for the best-performing architectures from the architecture experiments. For the unidirectional LSTM with Word2Vec embeddings, a learning rate of **0.01** yielded the highest test accuracy of **74.75%**,

while a dropout rate of **0.3** resulted in a test accuracy of **74.35%**. These results showed that a higher learning rate facilitated better convergence for this architecture. For the bidirectional LSTM, a learning rate of **0.001** provided the best test accuracy of **72.78%**, and a dropout rate of **0.5** significantly improved performance, achieving a test accuracy of **78.81%**. This suggests that moderate regularization through dropout was beneficial for the bidirectional model. Conversely, the stacked architectures showed a decline in performance across different hyperparameter settings, with the best test accuracy reaching only **60.22%**.

An evaluation was conducted on the impact of different pre-trained static embeddings on model performance. GloVe embeddings with dimensions of 100, 200, and 300, as well as FastText embeddings with a dimension of 300, were tested. The model utilizing GloVe embeddings of dimension **100** achieved a high test accuracy of **87.56%**, which was comparable to the performance with GloVe dimensions of 200 and 300. The FastText embeddings slightly outperformed GloVe, with a test accuracy of **87.86%**. These results suggest that both GloVe and FastText embeddings significantly enhance the model's ability to understand semantic relationships in the data, with minimal gains from increasing the embedding dimensions beyond 100.

The impact of fine-tuning pre-trained static embeddings during training was investigated to assess whether this would further improve performance. Using the same configurations as the previous experiment but allowing the embeddings to be updated, the model with GloVe embeddings of dimension **100** achieved a test accuracy of **85.28%**, which is slightly lower than when the embeddings were frozen. Similarly, the model with FastText embeddings obtained a test accuracy of **80.37%**, also lower than the non-fine-tuned counterpart. These results indicate that fine-tuning the embeddings did not lead to performance gains and, in fact, slightly reduced the model's accuracy. This suggests that the pre-trained embeddings already capture valuable semantic information, and altering them during training may disrupt this learned representation.

3. Final Architecture

Based on the results of our experiments, our final architecture utilizes pre-trained static word embeddings rather than learning embeddings during training. Specifically, we employed pre-trained embeddings from Word2Vec and GloVe, as these methods yielded the best performance. The Word2Vec model uses a unidirectional LSTM with one layer, a hidden dimension of 128, and an embedding dimension of 100. The embeddings were frozen. We set the learning rate to 0.01 and used a dropout rate of 0.5. This configuration achieved a test accuracy of **74.75%**. The GloVe model uses a similar unidirectional LSTM architecture. We maintained the embedding dimension at 100, set the learning rate to 0.001, and used the same dropout rate of 0.5. This setup resulted in a higher test accuracy of **87.56%**.

4. Replicating Results

All experiments were conducted on Kaggle using a GPU P100. The Kaggle notebook containing the full code and experiment details is available here: Kaggle Notebook. Additionally, the final architecture and configurations are provided in a Google Colab notebook for convenience: Google ColabNotebook. Both notebooks include all the necessary code, dependencies, and instructions to reproduce the experiments and achieve the reported results.

5. All Results

Architecture:

Experiment	Embeddings	Test Accuracy	Test Loss
Unidirectional	Learned Embeddings	0.5610	1.9428
Bidirectional		**0.6980**	**1.6113**
Stacked (2 Layers)		0.5663	2.5420
Stacked (3 Layers)		0.6418	2.5597
Unidirectional	Word2Vec	**0.7069**	**1.2399**
Bidirectional		0.6976	1.5862
Stacked (2 Layers)		0.6868	0.6868
Stacked (3 Layers)		0.6757	2.0653

Hyperparmeters:

Architecture	Hyper Parameter	Value	Test Accuracy	Test Loss
Unidirectional, Word2Vec	Learning Rate	0.001	0.7256	1.6809
		0.005	0.5953	2.2219
		0.01	**0.7475**	**1.7298**
	Dropout	**0.3**	**0.7435**	**1.4673**
		0.5	0.6686	1.5048
		0.7	0.7073	1.4947
Bidirectional, Word2Vec	Learning Rate	**0.001**	**0.7278**	**1.4149**
		0.005	0.6846	1.6998
		0.01	0.6677	2.1061
	Dropout	0.3	0.7203	1.4227
		0.5	**0.7881**	**1.1902**
		0.7	0.7650	1.3037
Stacked (2 Layer), Unidirectional, Word2Vec	Learning Rate	**0.001**	**0.5829**	**2.4963**
		0.005	0.5107	2.3448
		0.01	0.5742	2.6798
	Dropout	0.3	0.5914	2.4639
		0.5	0.5945	2.2250
		0.7	**0.6022**	**2.4807**

Embeddings:

Embeddings	Dimension	Test Accuracy	Test Loss
GloVe	**100**	**0.8756**	**0.5010**
	200	0.8756	0.7800
	300	0.8648	0.7573
FastText	**300**	**0.8786**	**0.6244**

Fine-Tuning:

Embeddings	Dimension	Test Accuracy	Test Loss
GloVe	**100**	**0.8528**	**0.928**
	200	0.8474	0.8077
	300	0.8488	0.9415
FastText	300	0.8037	0.9613