Supervised Classification Task for Scientific Discourse Concepts

Introduction

For the task of sentence-level classification, the significance of models selection and embedding training could not be neglected. The first assignment aims to develop a model capable of automatically identifying discourse concepts on sentence-level input. It also aims to experiment with the effects of combining different deep learning models with contextual and non-contextual (static) embedding vectors.

The procedure starts with dataset loading, turning textual data into a numeric representation. The selection of embedding types bifurcates the process. With respect to static embedding, the following process will be divided into two parts, including loading GloVe and initializing random embedding. The other option is to create context-sensitive token embedding using a pre-trained BERT model. The following step is doing experiments using three different models (including Deep Average Neural Network Model, Convolutional Model, and LSTM Model) with varying embedding vectors. These experimental results will be demonstrated in tabular format.

Experiments For Sentence-Level Classification

This section has conducted experiments exploring the effects of changing different language learning models, the sentence sequence length, embedding dimensions, and the implementation of the embedding techniques on the variation of accuracy rate. It is worthwhile to elaborate on the architecture layers for each model. The most basic model is the Deep Average Neural Network Model (DAN), which includes one embedding layer, one fully connected layer, and one average pooling layer. The second is the Convolutional Model, which consists of one embedding layer, at least two convolutional layers, and global pooling. This model enables the user to explore the effects on the result by using various filter sizes. In this assignment, the default filter sizes are three-gram and four-gram. The model applied by these two filter kernels will be concerted and passed into the dense layer. The latter is the LSTM model, which includes one embedding layer, one LSTM layer, and one global pooling layer.

1. Random Embedding

The experiment results yielded by models training on random embedding vectors are recognized as baselines. Generally, the accuracy scores obtained from these three models fluctuate around 0.56, implying that no distinction could be found regarding the model using random embedding. The DAN model performs relatively inferior accuracy score, which is reasonable since the default two simple dense layers are insufficient to capture the essential information. One possible solution is to increase the size of the network since a larger network will be able to express possible solution and is less likely to get stuck in a local minima. The accuracy rate for the CNN model (which is 0.5737) is slightly higher than the rest. It is interesting to find that CNN, even slightly, outperforms LSTM since LSTM has been proven its capability in processing long sequence sentences while CNN makes its name for feature detection.

	Precision	**Recall**	**Acc**	**AP**		**Precision**	**Recall**	**Acc**	**AP**
DAN - Train	0.673	0.590	0.768	0.649	**DAN - Val**	0.602	0.518	0.557	0.421
CNN - Train	**0.747**	**0.664**	**0.991**	**0.984**	**CNN - Val**	**0.608**	**0.524**	**0.574**	**0.436**
LSTM -Train	0.750	0.666	0.997	0.996	**LSTM - Val**	0.605	0.521	0.564	0.427

Best Practise	Model (CNN); Epochs (10); Optimizer (Adam); Learning Rate (0.006); Batch Size (50); Drop Out Rate (0.5); Sequence Length (64); Random Embedding; Embedding Size (300); **Accuracy: 0.574 on Validation Dataset**

Table 1 Model Performance with Hyper-parameters Tuning and Random Embedding

2. GloVe Embedding

The sole difference between this section and the last one is the implementation of embedding techniques. The results illustrated in table 2 are obtained from models trained on GloVe embedding. Similarly, the CNN model achieves the highest score (which is 0.620) in terms of accuracy, followed by LSTM (which has 0.617). The accuracy scores in this section are generally higher than those in the previous section, implying that GloVe embedding enhances models' capability in word representation, yet such an improvement is limited.

	Precision	Recall	Acc	AP		Precision	Recall	Acc	AP
DAN - Train	0.742	0.659	0.977	0.961	**DAN - Val**	0.621	0.537	0.613	0.473
CNN - Train	**0.749**	**0.665**	**0.996**	**0.993**	**CNN - Val**	**0.620**	**0.537**	**0.620**	**0.470**
LSTM -Train	0.748	0.665	0.997	0.994	**LSTM - Val**	0.617	0.535	0.617	0.464
Best Practise	Model (CNN); Epochs (10); Optimizer (Adam); Learning Rate (0.001); Batch Size (50); Drop Out Rate (0.5); Sequence Length (64); GloVe Embedding; Embedding Size (100); **Accuracy: 0.620 on Validation Dataset**								

Table 2 Model Performance with Hyper-parameters Tuning and GloVe Embedding

3. BERT Embedding

Unlike other embedding vectorization techniques like word2vec and GloVe, characterized as fixed-length embedding, BERT embedding could extract information from surrounding texts and convey a more accurate feature representation for the same word under different conditions. Assumably, a model with BERT embedding shall generally have better performance than the previous two types. However, the BERT models generally yield lower accuracy than those with GloVe and Random embedding. One possible explanation could be the failure to extract valuable information. The other could be lacking in fine-tuning.

	Precision	Recall	Acc	AP		Precision	Recall	Acc	AP
DAN - Train	0.716	0.633	0.899	0.834	**DAN - Val**	0.540	0.457	0.371	0.295
CNN - Train	**0.646**	**0.729**	**0.939**	**0.896**	**CNN - Val**	**0.555**	**0.472**	**0.416**	**0.319**
LSTM -Train	0.703	0.620	0.858	0.771	**LSTM - Val**	0.552	0.468	0.404	0.313
Best Practise	Model (CNN); Epochs (10); Optimizer (Adam); Learning Rate (0.001); Batch Size (50); Drop Out Rate (0.5); Sequence Length (64); BERT Embedding; Embedding Size (128); **Accuracy: 0.416 on Validation Dataset**								

Table 3 Model Performance with Hyper-parameters Tuning and BERT Embedding

Future Improvement

Firstly, the experiments are limited to no text preprocessing. Secondly, the tokenizer used in sections one and two is fundamental since it merely splits text by spaces, causing unnecessary out of vocabulary tokens (35,027) during the training process.