Assignment 1: Deep Learning Models for Sentiment Classification

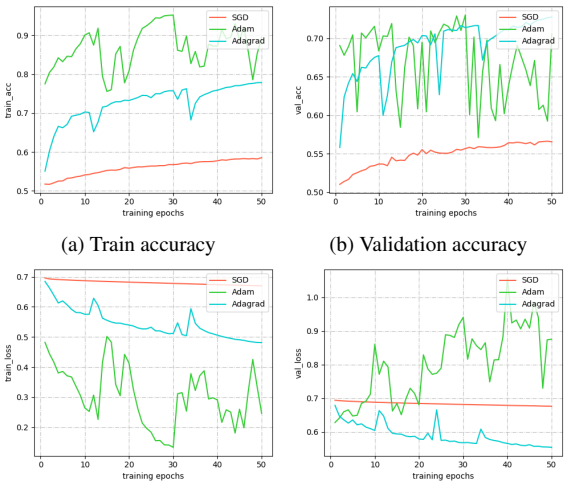
Zhang Siyue
siyue001@ntu.edu.sg

1. Task 1 (Abstract)

Following the code base provided, I re-implemented the Data Preparation, Model Building, Model Training, and Model Evaluation processes. In data preparation, the spaCy tokenizer was used to split the IMDB review sentences [1] into discrete tokens. A vocabulary was built with 25,000 words, while the rest words were replaced by the <unk> token and padding was represented by <pad>. The IMDB dataset was split into the train (17,500 samples), validation (7,500 samples), and test (25,000 samples) sets. The data batch was fixed at 1,500 sentence length for MLP and CNN models while the variable length was used for RNN models. BCE loss was adopted for measuring the error between the prediction and the ground truth label. Various optimizers were tested during the training, including SGD, Adam, and Adagrad for 50 epochs. Finally trained models are evaluated in the test dataset by loss and accuracy. The best test accuracy found in experiments was 84.86% by the 3-layer MLP (hidden dimensions 500, 300, and 200) with 77.71 million trainable parameters.

2. Task 2 & 3 (Optimizer)

The RNN model was trained with SGD, Adam, and Adagrad optimizers at the same learning rate (i.e., 1e-3) for 50 epochs. As shown in Fig.1c, the training loss of the SGD optimizer after 5, 10, 20, and 50 epochs is much higher than Adam and Adagrad, which implies a slower learning process. This is due to the fact that Adam and Adagrad adaptively adjust the learning rate (LR) based on past gradients (e.g., momentum). Because they increase LR when the gradient is too small, the convergence is accelerated. As the learning process is much slower for SGD and far from convergence, it has the lowest training and validation accuracy at epoch 50. Adam approaches the minimum loss quickly and has difficulty converging near the minimum due to its LR adaption behavior, which leads to an increase in validation loss over time as Fig. 1d and a large variance in validation accuracy as Fig. 1b. Based on the above observations, it can be concluded that 1e-3 LR is overvalued for the Adam optimizer and we can switch to the SGD optimizer near the end of the training to have a more stable convergence.



(a) Train accuracy (b) Validation accuracy

(c) Train loss (d) Validation loss

Figure 1. Comparison of different optimizers for RNN (SGD, Adam, Adagrad).

3. Task 4 (Embedding)

In this task, I investigated two word-embedding methods, i.e., randomly initialized embedding and pretrained Word2Vec embedding [2]. The controlled experiment was conducted with the following parameters: Adam optimizer, 50 epochs, batch size 32, embedding dimension 300, RNN model, and hidden dimension 512. A comparison has been made as Table 1. The weights of Word2Vec embedding were fixed during training due to the limited GPU memory.

It was observed that the Word2Vec embedding had a lower accuracy and a higher loss for this IMDB review sentiment classification task compared to the randomly initialized embedding, which was contrary to my expectation. Such result could be explained by the following reasons: the model with fixed pretrained weights has a smaller number of trainable parameters, thus lower model complexity

Method	# params	train_loss	train_acc (%)	valid_loss	valid_acc (%)	test_loss	test_acc (%)
Random	7.64m	**0.004**	**99.87**	1.107	**71.60**	0.681	**63.30**
Word2Vec	4.17m	0.659	60.19	**0.681**	56.46	**0.678**	56.41

Table 1. Performance comparison of word-embedding methods.

and capability; the IMDB reviews may have a different vocabulary from the one used by the Word2Vec model; the semantic captured by the Word2Vec embedding may not be effective for the target problem, a large amount of data could be required for domain adaptation; the default hyper parameters may not be effective for the Word2Vec embedding which requires to be correctly tuned. If weights of Word2Vec embedding are not frozen, the number of trainable parameters will rise to 900 millions from 4.17 millions, increasing the model complexity significantly.

4. Task 5 (Model)

Three types of models have been experimented with in this task, including MLP, CNN, and RNN. All models were trained by Adam optimizer, 50 epochs, LR 1e-3, and randomly initialized embedding size 100.

MLP models consist of various numbers and sizes of linear hidden layers between the embedding layer and output layer. MLP1 is a one-layer FFNN with a hidden dimension of 500. MLP2 has two layers with hidden dimensions 500 and 300, and MLP3 has three layers with dimensions 500, 300, and 200. After the embedding layer, the tensor is flattened into the dimension of [batch size, sentence length $\times$ embedding dimension], which leads to a large number of neurons in MLP models.

CNN models were built with three feature maps with different sizes (i.e., 1,2, and 3) following the approach of TextCNN in [3]. 2 feature map channels were used in the CNN1 model while 5 channels were set in the CNN2 model. The tensors after convolution were first max pooled and then concatenated into a 1-D vector before the output layer. Thanks to the convolutional layer, a substantial amount of trainable parameters were avoided.

LSTM models, as an advanced variant of RNN model, are capable of learning long-term dependencies, especially in sequence prediction problems. Apart from the hidden states, cell states are calculated as intermediate parameters in LSTM. The hidden layer dimension was 256. The bi-LSTM model refers to the bidirectional LSTM, which consists of a forward path and a backward path of LSTM units. Hidden states of two paths were concatenated before the output layer, which made bi-LSTM (3.23 million) have more trainable parameters than LSTM (2.87 million) and thus higher model complexity.

During the training of MLP and CNN models, batches of sentences were padded or cut to the same length (i.e., 1,500) before being fed to models. LSTM models received batches with variable length.

Model	# params	valid_loss	valid_acc (%)	test_loss	test_acc (%)
MLP1	77.50m	3.294	85.69	0.429	80.81
MLP2	77.65m	1.917	86.45	**0.379**	84.42
MLP3	77.71m	1.214	86.56	0.397	**84.86**
CNN1	2.501m	1.290	76.13	0.518	74.80
CNN2	2.503m	**0.949**	83.47	0.380	83.35
LSTM	2.87m	0.964	84.88	0.416	83.59
bi-LSTM	3.23m	0.976	**87.50**	0.394	84.14

Table 2. Test evaluation comparison of different models (MLP1: 1-layer FNN, MLP2: 2-layer FNN, MLP3: 3-layer FNN, CNN: 2-channel CNN, CNN2: 5-channel CNN, LSTM: unidirectional LSTM, LSTM2: bidirectional LSTM).

5. Discussion

According to the comparison made in Table 2, the model with the best test accuracy is MLP3, which has 77.71 million trainable parameters. The success of this model in the sentence classification task is mainly due to the high model complexity. Compared to other models, it took a longer time in each training epoch. It is worth noting that although CNN and RNN models have 20 to 30 times less trainable parameters, they achieve comparable accuracy to MLP models. It demonstrates the effectiveness of convolution and recurrent techniques, significantly improving the computational efficiency for model training and inference. Compared to CNN models, LSTM models had generally higher accuracy in validation and testing, which implies the advantage of memory mechanism over the convolution architecture for sequence data.

Taking a closer look at the training histories of different models in Fig. 2 and 3, the performance improvement of CNN and RNN models is much more significant than that of MLP models in the beginning of training. The training curves of CNN models were the most smooth while those of MLP models had significant variance. MLP models were much more difficult to converge compared to others. Overfitting was observed for all models in Fig. 3d as validation loss increased after 10 epochs. It could be due to the high LR and the high model complexity compared to the training data size. Decaying LR and regularization techniques could be used to mitigate the such issue.

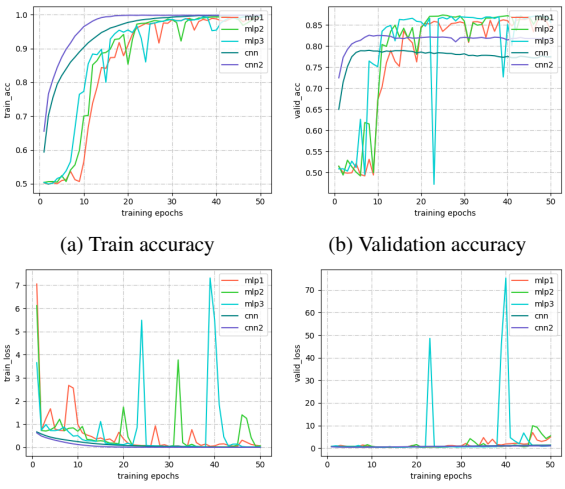
6. Conclusion

In conclusion, adaptively adjusted LR helps to accelerate the training process and improve model performance like Adam and Adagrad **optimizers**. But they might lead to oscillation and convergence issues. Therefore, techniques such as decaying LR are critical to be combined with Adam optimizer for a faster and smoother training curve.

Pretrained embedding could significantly improve the performance of NLP models by leveraging large amounts of data and complex language patterns that have been learned during the pretraining process. However, in this experiment, the pretrained embedding didn't outperform the random embedding. Various reasons could explain that: a smaller amount of trainable parameters used, different vo-

cabularies, semantics between target dataset and pretraining dataset, imperfect hyperparameters, etc. Fine-tuning is essential when applying pretrained models to new tasks.

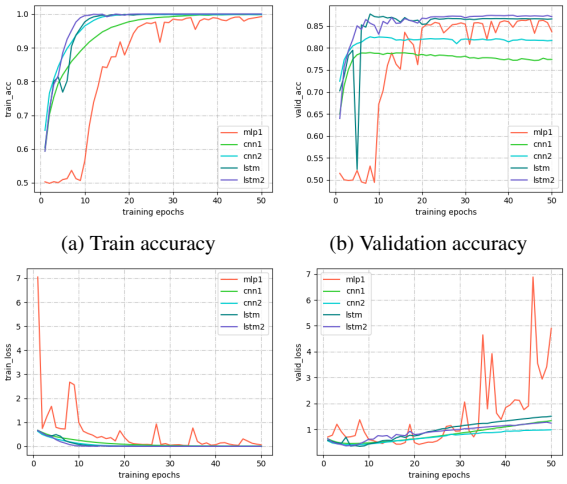
The **MLP models** demonstrated superior performance due to the high model complexity from concatenated word embeddings of words in the sentence. A large number of trainable parameters slows down the training speed. MLP models have obvious variance in the training process and difficulty converging within 50 epochs. Convolution mechanism of **CNN models** and memory mechanism of **LSTM models** showed great effectiveness in reducing required computation while maintaining high accuracy. The architecture and parameter-sharing mechanism greatly smooth the training curve. In contrast, MLPs do not have built-in mechanisms for detecting local patterns or spatial relationships in the input, which can lead to a more complex and noisy training process. CNN and RNN models are more efficient and effective in learning from complex input data such as images and sequences, and this can result in a smoother and more stable training process.



(a) Train accuracy (b) Validation accuracy

(c) Train loss (d) Validation loss

Figure 2. Comparison of different models (MLP1: 1-layer FNN, MLP2: 2-layer FNN, MLP3: 3-layer FNN, CNN: 2-channel CNN, CNN2: 5-channel CNN).



(a) Train accuracy (b) Validation accuracy

(c) Train loss (d) Validation loss

Figure 3. Comparison of different models (MLP1: 1-layer FNN, CNN: 2-channel CNN, CNN2: 5-channel CNN, LSTM: unidirectional LSTM, LSTM2: bidirectional LSTM).

References

[1] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. 1

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 1

[3] Yahui Chen. Convolutional neural network for sentence classification. Master's thesis, University of Waterloo, 2015. 2