Neural Network - Project 2 Report

Ida Novindasari - 2576861
Keyne Kassapa Oei - 2580667

February 2020

I. Task 1

a) Model: Brief description and analysis

The model architecture in task 1 is very simple (Fast Text Architecture). It consists of two main hidden layers which are the embedding bag layer and linear layer. The model uses 2-gram embedding and 'mean' mode to reduce the bag by averaging the values in the bag. Since the sentences are not equal, the gradient tensor is set to be a sparse tensor which will remove the '0' value in the vector. The second layer acted as the classifier using linear regression that put weights on the embedding dimension and reduces it to the number of expected features. The weights in these layers uses uniform distribution from range -0.5 to 0.5 as required.

For task 1, the criterion used is cross entropy loss function from pytorch that uses the softmax in the input of *log* to put the vector value of each data as the probability of the features. The optimizer used here are the stochastic gradient descent and scheduler. The scheduler decay the learning rate in each epoch by 0.9 of the original gradient.

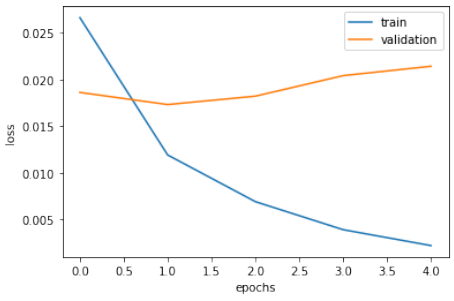

Figure 1: Task1 model loss

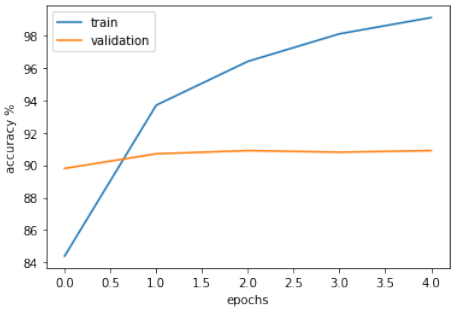

Figure 2: Task1 model accuracy

The performance of the task 1 model is fast and produce a good accuracy. Compared to the other methods we tried and proposed in task 2 and task

3, the model can obtain the same (or better) performance while being significantly faster. The model in task 1 uses the EmbeddingBag which is a huge advantage because the process is linear, thus, the overall process is quick compared to other techniques. With 5 epochs, task 1 model can achieve around 99% training accuracy, 91% validation accuracy and around 88% training accuracy. The objective function can be described as below [1]:
$-\frac{1}{N}\sum_{n=1}^{N} y_n log(f(BAx_n))$
where f is the linear classifier function, B and A is weight matrix from EmbeddingBag.

b) Question 3: Of course the predictions will be limited to the four class labels that your model is trained on. Can you somehow justify the labels that your model predicted now for the given text inputs?

- Music News: The model predicted the news as 'world' news. This is because there are lots of words, such as British, that may indicate it to the 'world' classification. We tried to run the model several times and it sometimes indicate this news as 'Business' news. Again, the reason for this might be from the fact that the weight of these words vector mainly in favor towards the 'Business' category compared to other categories. We find this quite reasonable since the music news we chose can be described as international music and how the music can be a success which can lean towards the business category.
- Game News: The model predicted the news as the 'Sci/Tec' news. This is because there are some of the Tech keywords, such as network, virtual, train, and other words, that might majorly impact the result.
- Film News: The model predicted the news as the 'World' news. This is because there are words that indicate the country name and lots of people name which may be the significant reason the result lean towards the 'world' news.

c) Question 4: Can you try to list down some improvements that you think would be able to improve the above model's performance?

- Adding more layer on the linear classification and activation functions to add non-linearity to the text classification problem. However, number of layers hypothetically is subject to the number of features.
- Adjust the hyperparams by using cross validation and early dropping to minimize the loss function with respect to the validation set.
- Instead of using the scheduler, use the adaptive learning rates with momentum to generalize the model.
- Modify the model to semi supervised model, therefore, can support to predict the class outside the known features.

- Make the n-gram with bigger value. Thus, the model can predict better with respect to the correlation of the words from n-gram.
- Other suggestions for improvements that change the architecture of task 1, such as adding CNN, RNN, and other techniques, are merely a gamble. Zolotov and Kung [5] analyse the fastText Classifier and they pointed how the text classification problem is still not well understood in terms of the fastText implementation.

II. Task 2

a) Model: Brief description and analysis

- Model Wang [4]

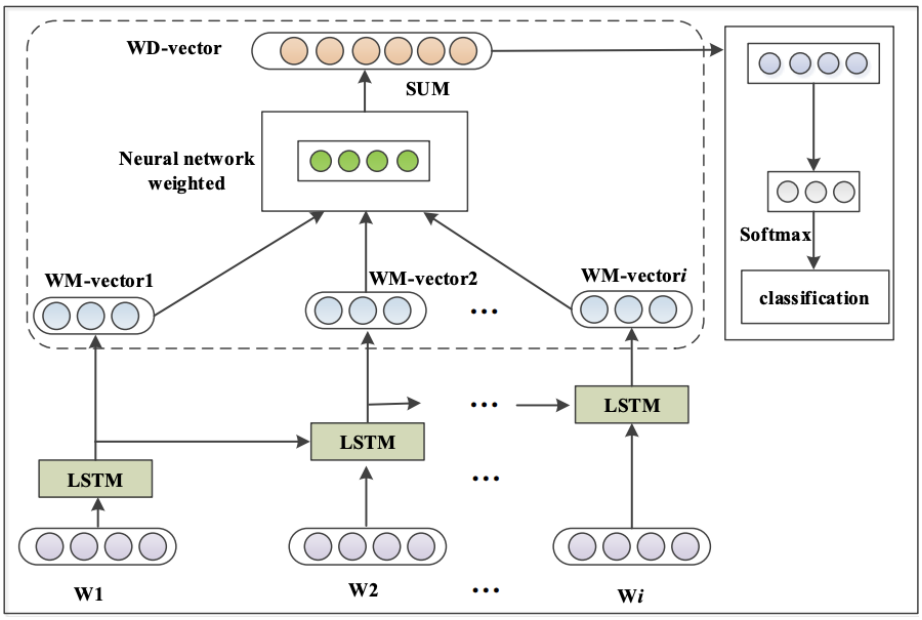

Figure 3: W-RNN Architecture [4]

The first model we try is from *Wang* which purposes a very simple modification to LSTM model for text classification.[4] The framework has 3 basic stages which are the LSTM part that acted to extract the text vocabulary and pre-text information, the W-RNN part that acted as the intermediate feature output vector, and the optimization part that uses softmax and cross entropy loss function. While the use of LSTM is good for text classification, *Wang* argues that it can't guarantee which effective information is recorded to the next stage. Therefore, *Wang* purposes the concept of W-RNN implementation that add more attention to the more important vocabulary information and also reduce attention to the less vocabulary information that control the impact of the text classification value. In our trials, we still keep the stochastic gradient descent optimization and scheduler with 2 n-grams.

By several trials with different hyperparameters, what *Wang* purposes can't outperform the model that is purposed in task 1. The paper indicate that the architecture can obtain the 99% accuracy from 10 iterations. However, from our implementation, *Wang* model

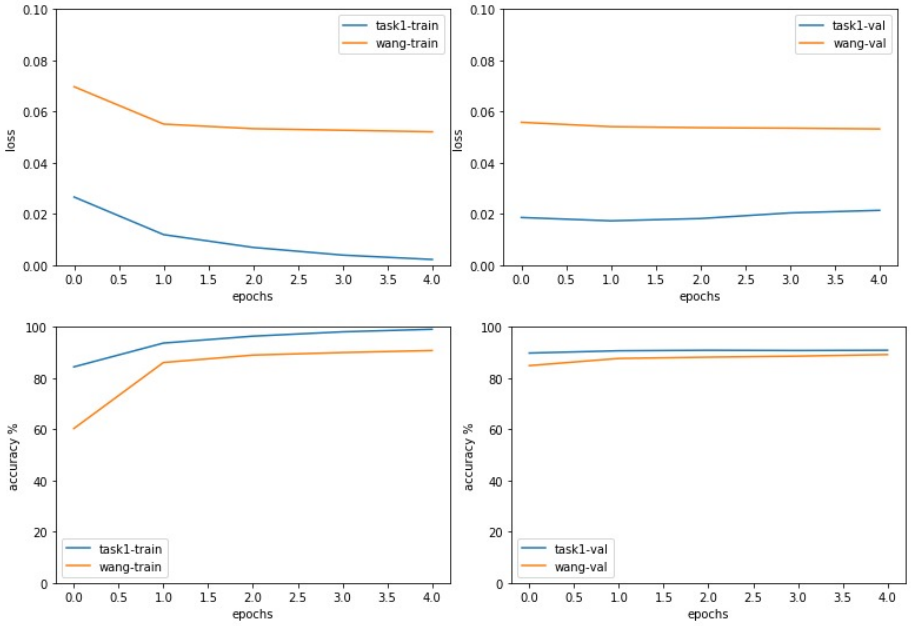

Figure 4: Wang accuracy and loss on training and validation set compared to the task 1 model [4]

can only achieve around 93% of training accuracy and 89% of validation accuracy. The accuracy from test set from *Wang* can achieve around 89% which may or may not out perform the test set accuracy from task 1 model depending on the weight initialisation.

In the paper, *Wang* mentioned that the model has several shortcomings which we also experience for AG NEWS dataset model. One of the most noticeable drawback is that the model can't generalized while maintaining a good training accuracy. Another drawback reason may rely on how the forward hidden state with given weight can't represent the important words well enough like the model in task 1 or model like what *Li* [2] purposes. Thus, some of the important words such as country names to classify the news to 'World' category can't be recognized as well as the other model implementations with CNN and other techniques.

- Model Li [2]

  The second model we try is from *Li*. [2] *Li* purposes a sentiment feature enhanced deep neural network with GRU and CNN mechanism. *Li* argues that the combination of GRU and CNN enhance and maximize the representation of the text structure. *Li* pointed that CNN performs poorly in sequential correlation information but can help what GRU lacks to extract the local features. As shown in

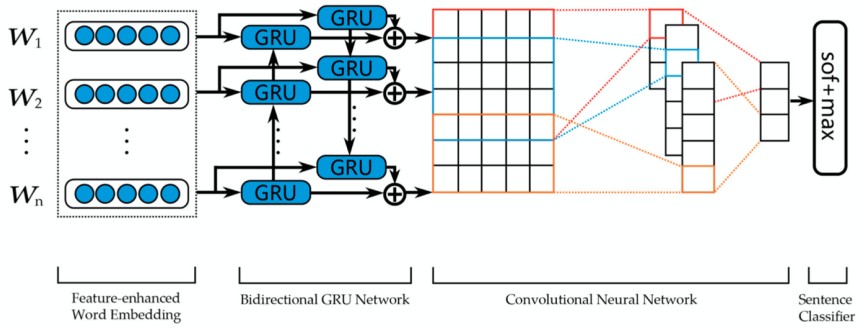

Figure 5: The architecture of the proposed sentiment-feature-enhanced deep neural network (SDNN) model. [2]

the Figure 5, the model that *Li* purposed consist of 4 major parts which are the feature enhanced word embedding, bidirectional GRU network, CNN and sentence classifier using softmax. In our trials, we still keep the stochastic gradient descent optimization and scheduler with 2 n-grams.

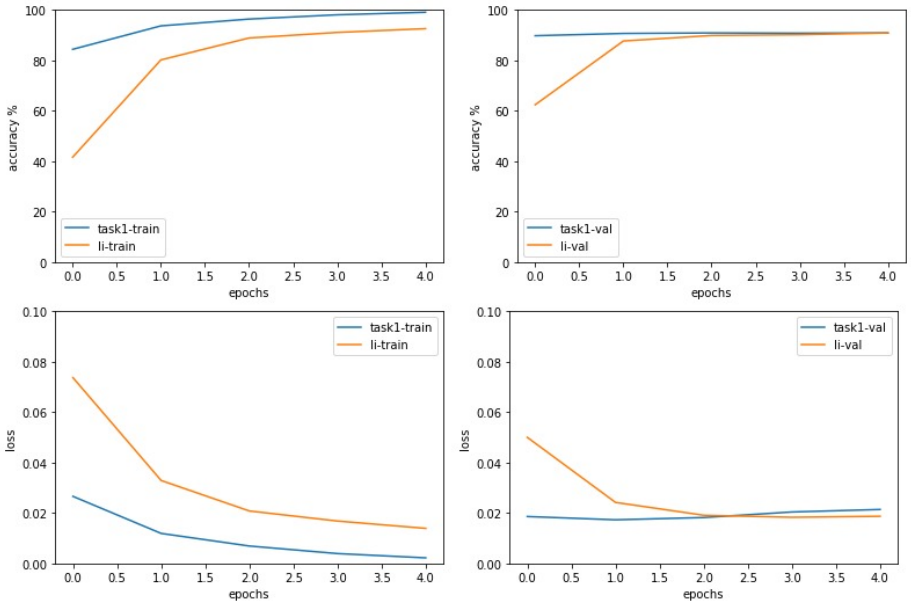

Figure 6: Li accuracy and loss on training and validation set compared to task 1

What *Li* purposes **outperform** the performance compared to task

1 by only a little difference on validation and test set. It does **not outperform** the training set still. The reason might be because the implementation of all the hidden state in the CNN fail to highlight the word sequence in the sentence. Therefore, we modify this model to be more suitable for our text classification problem, which we will discuss on the next point. One of the main drawback from this model is the long training time because the use of GRU with CNN is computationally very expensive. While we can train the model in task 1 around 1.5 minutes per epoch, the model that *Li* purposes can take around 20-30 minutes per epoch on AG News dataset.

- Model Modified Li [2]

  Since the model in *Li* without pre-trained embedding can't outperform the task 1 model, we modify some of the *Li* architecture to be more suitable. In the modified *Li* model, we only use the last output of the bi-GRU process. Then, we put the forward and hidden state of the bi-GRU output into our channel in CNN. The purpose of this modification is to highlight the result of sequential (end of hidden state) rather than to put the attention to each of the hidden state. In our trials, we still keep the stochastic gradient descent optimization and scheduler with 2 n-grams.

  The performance of this modified *Li* is better than what the original model *Li*. The results show that the model can achieve around 95% on training dataset, 91% on validation dataset and 90% on test dataset after 5 epochs. Again, although we **outperform** the validation and test accuracy, we still can't beat the training accuracy. To beat the task 1 model training accuracy (99%), we need to select our hyperparameters carefully and train with more epochs. However, like the original model *Li*, this model also suffer from expensive computational, therefore, take a very long time to be trained.

- Model Li with GloVe [2]

  In the architecture of Model *Li*, they used 200-dimensional GloVe vectors pre-trained by Pennington et al. [3]. We used pre-trained GloVe from pytorch library that is trained on 6B-tokens with 50-dimension. For the words that are not found in the dictionary, we also used the same value as the paper with uniform value with range -0.1 and 0.1. The optimizer that are used is Adam with default learning rate of 10^{-2}. For the convolutional layer, we used the same filter windows 3, 4, and 5 with 8 feature maps each.

  The result by using GloVe shows a higher performance in the test set. By using pre-trained embedding, we already have the converge vector for our embedding. Thus in the first epoch, we already got 89% accuracy for the training set and 90% in the validation set. For the test set, we achieved 91.4% accuracy by only training for 5 epochs. The test set performance **outperform** the model task 1 test set accuracy by 5%. One of the main advantage from using GloVe is

the faster training because it convert the embedding value with pre-trained word to vector, thus, there is no need to update the weight for the word embedding. The model *Li* with word embedding takes around 40 minutes to train per epoch, whilst with GloVe only take around 15 minutes to train per epoch.

We also try to continue training the word embedding pre-trained from GloVe. As seen in Table 1, by continuing the word embedding training until 5 epochs, the result shows better performance both in each epoch and in the final test accuracy. The result also increases the performance on the training data during the training process.

	pre-trained GloVe embedding	**continue GloVe embedding training**
epoch 1 [train/val]	89.1 / 90.1	90.3 / 92.1
epoch 2 [train/val]	91.1 / 90.6	94.0 / 92.9
epoch 3 [train/val]	91.7 / 91.0	96.0 / 92.9
epoch 4 [train/val]	92.2 / 90.8	97.4 / 92.4
epoch 5 [train/val]	92.6 / 91.3	98.4 / 91.7
test accuracy	91.4	**91.5**

Table 1: Comparison result of continuing training the word embedding vs not continuing training the word embedding.

- Conclusion

  The comparison from all models above are visualized in the Table 2. It can be concluded that, by using 5 epochs, the Model *Li* with GloVe can **outperform** the other models. All the models are generally achieve better (or same) performance with task 1 model on validation and test set. However, since the model generalizes, the training performance can't beat the task 1 model training accuracy. (99%)

	Wang	**Li w/o GloVe**	**Modified Li**	**Li GloVe**
epoch 1 [train / val]	60.3 / 84.9	41.6 / 62.4	76.0 / 86.6	90.3 / 92.1
epoch 2 [train / val]	86.1 / 87.7	80.2 / 87.7	90.4 / 90.2	94.0 / 92.9
epoch 3 [train / val]	89.0 / 88.2	88.9 / 89.9	92.7 / 91.1	96.0 / 92.9
epoch 4 [train / val]	90.0 / 88.6	91.1 / 90.2	94.2 / 91.0	97.4 / 92.4
epoch 5 [train / val]	90.8 / 89.2	92.6 / 90.9	95.7 / 90.8	98.4 / 91.7
test accuracy	89.0	89.0	90.1	**91.5**

Table 2: Performance comparison between Wang, Li, Modified Li and Li GloVe models

III. Task 3

a) Model: Brief description and analysis

To try outperform the previous task models, we tries several models such as below.

- Weighted bi-GRU & CNN

  What *Li* purposes in the previous task mentioned the use of both bi-GRU and CNN to maximize the potential of the text representation. Since the model uses the CNN after the implementation of bi-GRU, it might diminishes the importance of sequential that bi-GRU has produced. As *Li et al.* [2] mentioned that GRU lacks to extract local features, we want to capture the local features itself from the original embedding. Therefore, we propose the use of CNN and bi-GRU in parallel, then put weights on element of the CNN and bi-GRU output. The other reason we use CNN and bi-GRU parallelly because CNN might generate more important feature space from the original dataset compared to the bi-GRU elements as the CNN input like what *Li* has proposed before.

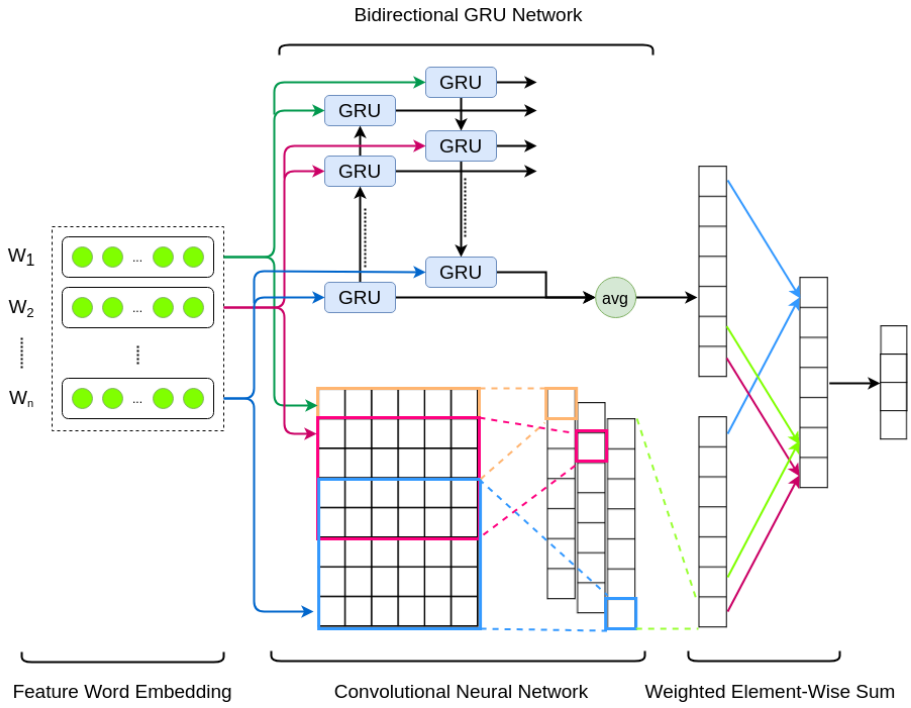

Figure 7: Weighted bi-GRU & CNN Architecture

There are 3 major parts of our new model; the feature word embed-

ding, CNN part, and weighted element-wise sum part. The architecture of our new model is visualized in Figure 7. We use the same embedding dim like the task 1 model which is 32 dimensional. The bi-GRU in our model will take the output of embedding and generate the new 8 (one per four of the embedding dimensional) hidden states at the end of the sequence. Since we knew from the modified *Li* model that the use of the end of sequence hidden state generates better performance, the model only uses the last hidden state. Another advantage of this is that it will highlight the sequence of words in the sentence. The forward and backward output of bi-GRU are averaged. CNN will take the output of embedding and generate output matrix of 3 kernels with 8 channels. Then, the model will average the 3 matrix from CNN so that it yields the same feature space dimensional like the bi-GRU output. Afterward, to combine bi-GRU and CNN, we put weights on both techniques on **the same feature space index. The weights that are applied here are trained to put the favor towards either bi-GRU or CNN.** At the end of the architecture, we applied linear layer to reduce the 8 feature space to 4 as the output layer.

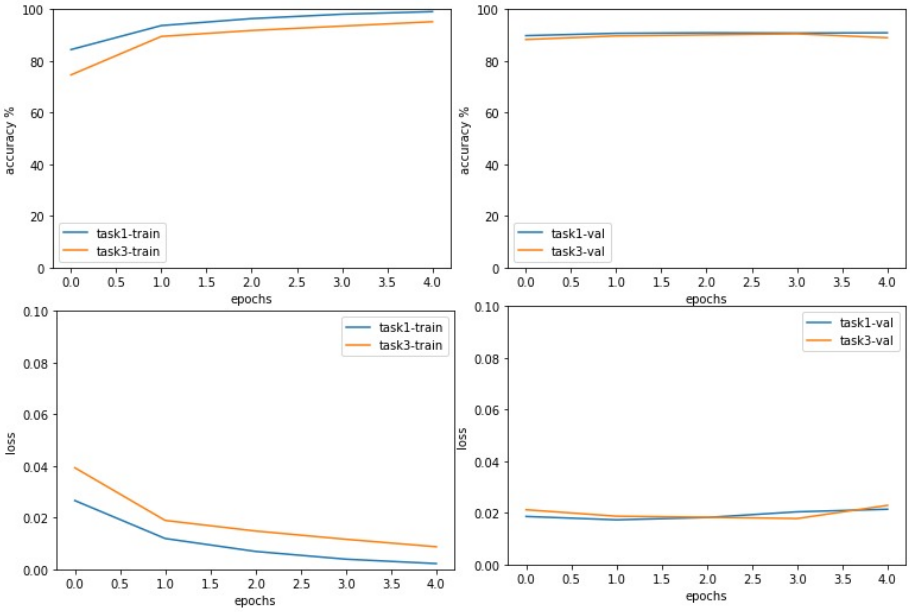

Figure 8: Weighted bi-GRU & CNN accuracy and loss on training and validation set compared to task 1 model

The accuracy of our model can achieve 95% on training dataset, 89% on validation dataset, and around 90% on test dataset. The

performance of our new model is close to task 1 performance, as seen in the Figure 8. It **outperforms** the test accuracy but still can **not outperform** the training and validation set. Compared to *Li*, the training performance surprisingly is better by around 1-3% and the training dataset is better by around 1%. In general, our model can perform better in test and training dataset but performs quite poorly on validation dataset. Our model still suffer from expensive computation. However, our model can train the bi-GRU and CNN in separate machine of CPUs/GPUs, unlike *Li* and *Wang* models. Since we don't have two parallel CPU/GPU to prove this theory, it presumably will perform faster.

- Pre-trained with weighted GRU & Conv

  Since we know that the performance with the pre-trained GloVe will yield better performance, we want to try our new model with GloVe to increase the performance. The GloVe embedding that is used is the same as the previous model with *Li* architecture.

  With GloVe, we can achieve higher accuracy on train and test dataset. The model can achieve 93% training accuracy, 92% validation accuracy and 92% test accuracy. The overall result has higher accuracy around 5% more than the model without pre-trained GloVe. We need more epoch to out perform the task 1 training accuracy (99%). **So far, this model is our best model with very high test and validation accuracy as well with the training accuracy that is very close to 99%.**

	2-grams without GloVe	**1-gram GloVe only pre-trained embedding**	**1-gram GloVe continue embedding training**
epoch 1 [train / val]	74.6 / 88.3	89.5 /90.8	90.9 / 92.3
epoch 2 [train / val]	89.5 / 89.7	91.4 / 91.6	94.5 / 93.3
epoch 3 [train / val]	91.8 / 90.1	92.3 / 91.8	96.5 / 93.1
epoch 4 [train / val]	93.5 / 90.5	92.8 / 92.0	97.8 / 92.5
epoch 5 [train / val]	95.2 / 89.0	93.2 / 92.2	98.8 / 92.3
test accuracy	89.9	91.7	**91.8**

Table 3: Comparison of the proposed model with 2-grams without GloVe and 1-gram with GloVe. The first GloVe version is not doing any training on the word embedding, while the other version is continue training on the word embedding.

References

[1] Armand Joulin et al. *Bag of Tricks for Efficient Text Classification*. 2016. arXiv: 1607.01759 [cs.CL].

[2] Li, Liu, and Zhang. "An Improved Approach for Text Sentiment Classification Based on a Deep Neural Network via a Sentiment Attention Mechanism". In: *Future Internet* 11 (Apr. 2019), p. 96. DOI: 10.3390/fi11040096.

[3] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[4] Dan Wang, Jibing Gong, and Yaxi Song. *W-RNN: News text classification based on a Weighted RNN*. 2019. arXiv: 1909.13077 [cs.IR].

[5] Vladimir Zolotov and David Kung. "Analysis and Optimization of fastText Linear Text Classifier". In: *ArXiv* abs/1702.05531 (2017).