CS6375 Assignemnt 1

https://github.com/g-h-chen/CS6375_Assignemnt1

Guiming Hardy Chen
ghc240000

---

1 Introduction and Data

Introduction In this project, we implement Feed Forward Neural Network and Recurrent Neural Network (RNN) to perform sentiment analysis, which is a basic classification task in NLP. We train both models with different hidden dimensions and find that overfitting issue exists for both models, indicating that the task is relatively simple. Finally, we propose that this issue may be solved by adding regularization terms or dropout functions. Meanwhile, the model performance can be furthered improved by adding positional information into word embedding to replace the current Bag-of-Word classification.

Dataset Statistics We summarize our dataset statistics in Table 1, including sample counts and average length of samples for each split.

Split	#samples	Avg. Length (std)
Training	16,000	124.7 ± 114.2
Validation	800	140.4 ± 117.9
Test	800	109.8 ± 96.4

Table 1: Dataset statistics.

2 Implementations

2.1 FFNN

Implementation Details Below is the code snippet for the `forward` method:

```
def forward(self, input_vector):
    # [to fill] obtain first hidden layer representation
    out = self.activation(self.W1(input_vector))

    # [to fill] obtain output layer representation
    out = self.W2(out)

    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(out)
    return predicted_vector
```

Let V be the vocabulary size, D be the hidden dimension of the model, C be the number of classes. The corresponding mathematical operations for each line are detailed as follow:

1. Op1: $out_1 = ReLU(W_1 x)$, where $x \in R^{V \times 1}$ is the `input_vector`, $W_1 \in R^{D \times V}$ is the first transition matrix.
2. Op2: $out_2 = W_2 out_1$, where $out_1 \in R^{D \times 1}$ is the output of Op1, $W_2 \in R^{V \times C}$ is the second transition matrix that maps hidden states to the dimension of output classes.
3. Op3: $out_3 = LogSoftmax(out_2)$, where $out_3 \in R^C$ is the distribution vector over C classes.

Libraries and Tools For debugging, we used `pdb`, which is a useful tool for adding breakpoints anywhere in the code. The program is able to pause at anywhere and we can do interactive debugging in the terminal.

Understanding of Codes We provide a flow description of the code as follows:

1. Loading data: The data are laded from a json file with each sample being a `(text, label)` tuple. Then a vocabulary set is built based on the training set and the unit in the vocab is "word" rather than "tokens", which is more frequently used in NLP tasks. An index set is built on top of the vocabulary for faster indexing.

2. Vectorization: To obtain the vectorized representation of each sample, we initialize a zero-vector, essentially a word counter, and iterate over all words in a sample. The word counter is of length V, with each entry being the number of appearances of the corresponding word in the sample. Note that the way of vectorization is generic, because it solely treats a sample as bag-of-word (cite), and ignores the positional information between words which can be vital for classification tasks.

3. Training: Stochastic gradient descent (SGD) with learning rate 0.01 and momentum 0.9 is used for training. `NLLLoss`, together with `nn.LogSoftmax`, is used as the loss function. This is equivalent to using `nn.CrossEntropyLoss` alone. The code manually batches the samples, and perform back-propagation when after the computation is done for a batch. The process is performed in a sample-by-sample fashion, which does not fully utilize the parallel feature provided by PyTorch.

4. Validation: The validation is performed at the end of each epoch. It computes the validation loss over the validation set.

2.2 RNN

Implementation Details Below is the code snippet for the `forward` method:

```
def forward(self, inputs):
    '''
    inputs: (seqlen, bsz=1, dim=50)
    '''
    # [to fill] obtain hidden layer representation
    output, hidden = self.rnn(inputs)
    # output: (seqlen, bsz, hidden_dim)
    # hidden: (n_layer, bsz=1, hidden_dim)
    # output[-1] == hidden[-1]

    # [to fill] obtain output layer representations
    out = self.W(output)

    # [to fill] sum over output
    out = out.sum(0).squeeze(0)

    # [to fill] obtain probability dist.
    out = self.softmax(out)
    return out
```

Let K be the input dimension, D be the hidden dimension of the model, C be the number of classes, and L_x be the sequence length of the sample x.

The corresponding mathematical operations for each line are detailed as follow:

1. Op1: $out_1, \_ = RNN(x)$, where x with shape `(`L_x`,1,`K`)` is the `input_vector`, and out_1 with shape `(`L_x`,1,`D`)` is the last hidden states for each token.

2. Op2: $out_2 = Wout_1$, where $W \in R^{D \times C}$ is the transition matrix that maps hidden state dimension to the dimension of output classes, and out_2 with shape `(`L_x`,1,`C`)` is the tensor of logits for each token.

3. Op3: $out_3 = out_2.sum(0).squeeze(0)$, where $.sum(0)$ sums the logits for all tokens, and $.squeeze(0)$ squeezes the tensor from 2-D to 1-D.

4. Op4: $out_4 = LogSoftmax(out_3)$, where $out_3 \in R^C$ is the distribution vector over C classes.

Libraries and Tools For debugging, we used `pdb`, which is a useful tool for adding breakpoints anywhere in the code. The program is able to pause at anywhere and we can do interactive debugging in the terminal.

Understanding of Codes We provide a flow description of the code as follows:

1. Loading data: Same as in FFNN.

2. Vectorization: To obtain the vectorized representation of each sample, we first tokenize each sample into multiple words, which are then converted to their corresponding vector representations through a lookup table (`word_embedding.pkl`).

3. Training: Adam [1] is used as optimizer. In the original template, the training terminates when training accuracy increases and validation accuracy decreases (which is a sign of overfitting). However, we find that the stopping criteria is too sensitive. Therefore, we train all models with 25 epochs and plot the learning curve.

4. Validation: The model is validated at the end of each epoch.

3 Experiments and Results

3.1 Evaluations

For both models, we use two metrics to evaluate model performance:

1. **Loss**: It is the standard cross-entropy loss used for classification task. The validation loss is for preventing overfitting.

2. **Accuracy**: A more straightforward performance measurement.

3.2 Results

FFNN The default hidden dimension is 10. Additionally, we run experiments on dimensions 128, 512, 1024, 2048. We plot the epoch losses in Fig. 1 and summarize the best validation accuracies in Fig. 2.

The loss and accuracy curves are consistent, where they should be negatively correlated. As the number of dimension increases, the training loss becomes lower, but the validation loss becomes much higher, which is a sign of overfitting. Meanwhile, we observe that the validation accuracy curves are almost flat compared to the ascending training accuracy as training proceeds. This indicates that FFNN model may fail to generalize from what is learned on training set to validation set, suggesting that a better model should be adopted.

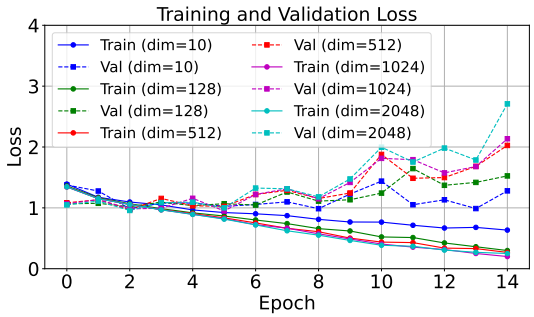

Figure 1: FFNN loss plot.

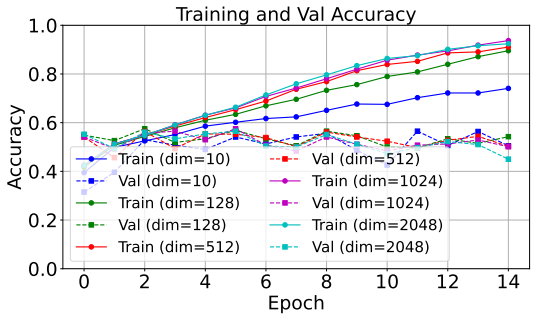

Figure 2: FFNN accuracy plot.

RNN The default hidden dimension is 32. Additionally, we run experiments on dimensions 8, 16, 128, 256. We plot the epoch losses in Fig. 3 and the best validation accuracies in Fig. 4.

The loss and accuracy curves are consistent, where they should be negatively correlated. As the number of dimension increases, the training loss becomes lower, but the validation loss becomes much higher, which is a sign of overfitting. Meanwhile, we observe that the validation accuracy curves are almost flat compared to the ascending training accuracy as training proceeds. This indicates that FFNN model may fail to generalize from what is learned on training set to validation set, suggesting that a better model should be adopted.

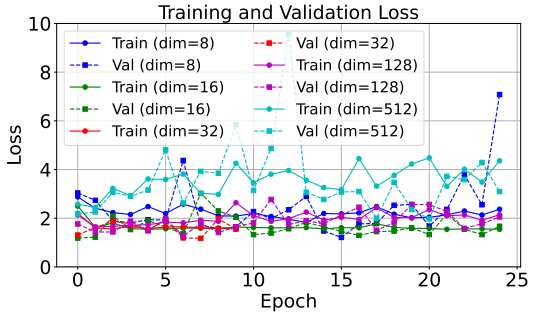

Figure 3: RNN loss plot.

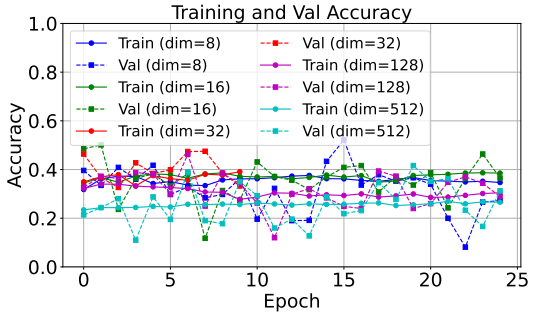

Figure 4: RNN accuracy plot.

4 Analysis

Learning Curves The learning curves for both FFNN and RNN are plotted above.

Potential improvement on the system Based on what we observe in the experiments above, we propose several possible remedies to improve the system. Firstly, **adding dropout/regularization to avoid overfitting**, which are two commonly used techniques. Secondly, **adding positional embedding**. By doing so, the model is aware of the position of each word (*e.g.*, as in [2]) rather than solely performing Bag-of-Word [3] learning.

5 Conclusion

5.1 Contribution

Guiming Hardy Chen completed codes and this report on his own, and is responsible for all results.

5.2 Suggestions

The validation set is class-imbalance. The Val split only has class 0,1,2, while test split only has class 2,3,4. Monitoring the performance on such an imbalanced validation set may not provide optimal results as expected.

Clearer instructions should be provided. In the `forward` function of RNN, there's a hint saying "[to fill] sum over output". It is not clear whether we should sum over the `output` or the `hidden` variable. Summing over the former means aggregating information from **all token representations in the last layer**, while summing over the latter means aggregating information from **the last token representations in all layers**.

To have a better understanding of the aggregation strategy, the latter summing strategy is implemented as well, yielding slightly lower performance than the former. This implies that the recurrent nature of RNN may posses forgetting problem across the temporal dimension. On the other hand, features from shallower layers may not contribute as much as those from the last layer to classification performance. Therefore, it is beneficial to aggregate each token from the last layer to perform the final classification.

References

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[2] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[3] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International journal of machine learning and cybernetics*, vol. 1, pp. 43–52, 2010.