CS539 - NLP - Homework 2

Nuttaree Busarapongpanich

1 Demystifying Recurrent Neural Networks

1.1 Deriving LSTM Gradients for Univariate Case

TASK 1.1

The gradient for h_t w.r.t. c_t:

$$\frac{\delta h_t}{\delta c_t} = o_t * (1 - tanh^2(c_t)) \tag{1}$$

The gradient for c_t w.r.t. i_t:

$$\frac{\delta c_t}{\delta i_t} = g_t \tag{2}$$

The gradient for a i_t w.r.t. w_{ix}:

$$\frac{\delta i_t}{\delta w_{ix}} = \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i) * (1 - \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i)) * x_t \tag{3}$$

The gradient for a i_t w.r.t. b_i:

$$\frac{\delta i_t}{\delta b_i} = \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i) * (1 - \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i)) \tag{4}$$

The gradient for h_t w.r.t. w_{ix}. According to chain rule:

$$\frac{\delta h_t}{\delta w_{ix}} = \frac{\delta h_t}{\delta c_t} * \frac{\delta c_t}{\delta i_t} * \frac{\delta i_t}{\delta w_{ix}} \tag{5}$$

Substitute (5) by (1), (2), (3):

$$\frac{\delta h_t}{\delta w_{ix}} = o_t * (1 - tanh^2(c_t)) * g_t * \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i) * (1 - \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i)) * x_t \tag{6}$$

Next, we want to find the gradient for h_t w.r.t. b_i. According to chain rule:

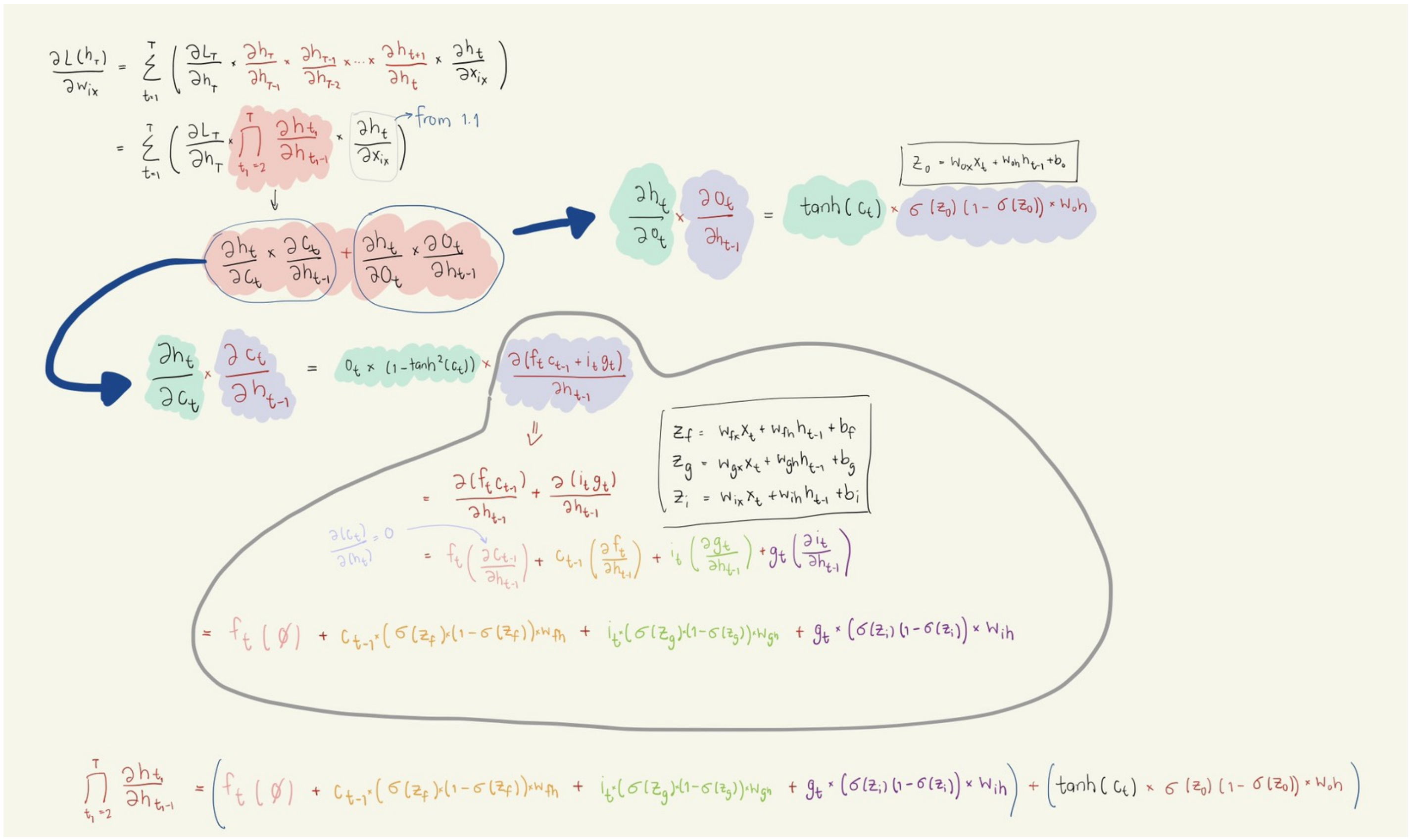
$$\frac{\delta h_t}{\delta b_i} = \frac{\delta h_t}{\delta c_t} * \frac{\delta c_t}{\delta i_t} * \frac{\delta i_t}{\delta b_i} \tag{7}$$

Substitute (7) by (1), (2), (4):

$$\frac{\delta h_t}{\delta b_i} = o_t * (1 - tanh^2(c_t)) * g_t * \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i) * (1 - \sigma(w_{ix}x_t + w_{ih}h_{t-1} + b_i)) \tag{8}$$

TASK 1.1 extra credit

The gradient for a loss w.r.t. i_t



$$\frac{\partial L(h_T)}{\partial w_{ix}} = \sum_{t=1}^{T}\left(\frac{\partial L_T}{\partial h_T}\times\frac{\partial h_T}{\partial h_{T-1}}\times\frac{\partial h_{T-1}}{\partial h_{T-2}}\times\dots\times\frac{\partial h_{t+1}}{\partial h_t}\times\frac{\partial h_t}{\partial x_{ix}}\right)$$

$$= \sum_{t=1}^{T}\left(\frac{\partial L_T}{\partial h_T}\times\prod_{t_1=2}^{T}\frac{\partial h_{t_1}}{\partial h_{t_1-1}}\times\boxed{\frac{\partial h_t}{\partial x_{ix}}}\right) \text{ from 1.1}$$

$$\frac{\partial h_t}{\partial c_t}\times\frac{\partial c_t}{\partial h_{t-1}} + \frac{\partial h_t}{\partial o_t}\times\frac{\partial o_t}{\partial h_{t-1}}$$

$$\boxed{z_o = W_{ox}x_t + W_{oh}h_{t-1} + b_o}$$

$$\frac{\partial h_t}{\partial o_t}\times\frac{\partial o_t}{\partial h_{t-1}} = \tanh(c_t)\times\sigma(z_o)(1-\sigma(z_o))\times W_{oh}$$

$$\frac{\partial h_t}{\partial c_t}\times\frac{\partial c_t}{\partial h_{t-1}} = o_t\times(1-\tanh^2(c_t))\times\frac{\partial(f_t c_{t-1} + i_t g_t)}{\partial h_{t-1}}$$

$$\boxed{\begin{aligned} z_f &= W_{fx}x_t + W_{fh}h_{t-1} + b_f \\ z_g &= W_{gx}x_t + W_{gh}h_{t-1} + b_g \\ z_i &= W_{ix}x_t + W_{ih}h_{t-1} + b_i \end{aligned}}$$

$$= \frac{\partial(f_t c_{t-1})}{\partial h_{t-1}} + \frac{\partial(i_t g_t)}{\partial h_{t-1}}$$

$$\frac{\partial(c_t)}{\partial(h_t)} = 0$$

$$= f_t\left(\frac{\partial c_{t-1}}{\partial h_{t-1}}\right) + c_{t-1}\left(\frac{\partial f_t}{\partial h_{t-1}}\right) + i_t\left(\frac{\partial g_t}{\partial h_{t-1}}\right) + g_t\left(\frac{\partial i_t}{\partial h_{t-1}}\right)$$

$$= f_t(\emptyset) + c_{t-1}\times(\sigma(z_f)\times(1-\sigma(z_f))\times W_{fh} + i_t\times(\sigma(z_g)\times(1-\sigma(z_g))\times W_{gh} + g_t\times(\sigma(z_i)(1-\sigma(z_i))\times W_{ih}$$

$$\prod_{t_1=2}^{T}\frac{\partial h_{t_1}}{\partial h_{t_1-1}} = \left(f_t(\emptyset) + c_{t-1}\times(\sigma(z_f)\times(1-\sigma(z_f))\times W_{fh} + i_t\times(\sigma(z_g)\times(1-\sigma(z_g))\times W_{gh} + g_t\times(\sigma(z_i)(1-\sigma(z_i))\times W_{ih}\right) + \left(\tanh(c_t)\times\sigma(z_o)(1-\sigma(z_o))\times W_{oh}\right)$$

1.2 Hand Designing A Network for Parity

TASK 1.2

One of the weights and biases for the univariate LSTM that can solve parity of binary strings of arbitrary length.

$$\begin{aligned}
w_{ix} &= 40 \\
w_{ih} &= 40 \\
b_i &= -20 \\
w_{fx} &= 0 \\
w_{fh} &= 0 \\
b_f &= -50 \\
w_{ox} &= -40 \\
w_{oh} &= -40 \\
b_o &= 60 \\
w_{gx} &= 0 \\
w_{gh} &= 0 \\
b_g &= 50
\end{aligned}$$

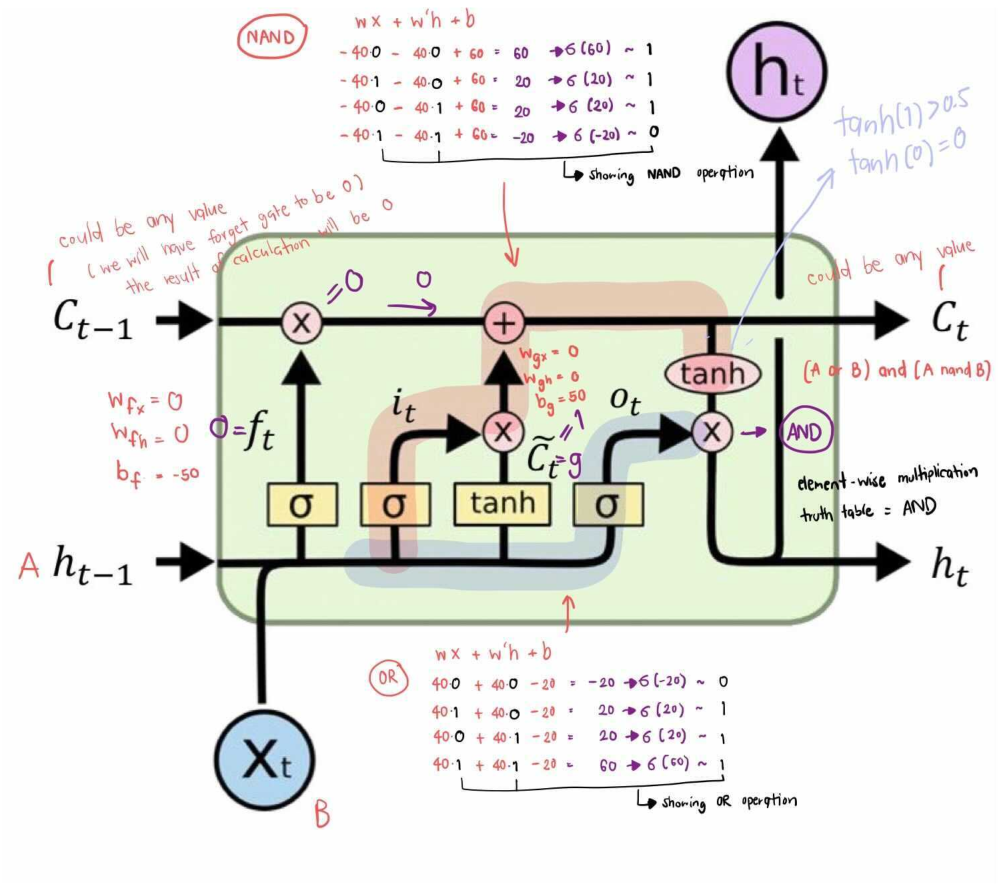

NAND
w x + w'h + b
-40·0 - 40·0 + 60 = 60 → σ(60) ~ 1
-40·1 - 40·0 + 60 = 20 → σ(20) ~ 1
-40·0 - 40·1 + 60 = 20 → σ(20) ~ 1
-40·1 - 40·1 + 60 = -20 → σ(-20) ~ 0
Showing NAND operation
h_t
tanh(1) > 0.5
tanh(0) = 0
could be any value
(we will have forget gate to be 0)
the result of calculation will be 0
C_{t-1}
=0
0
C_t
could be any value
(A or B) and (A nand B)
tanh
W_fx = 0
W_fh = 0
b_f = -50
0 = f_t
i_t
W_gx = 0
W_gh = 0
b_g = 50
o_t
C̃_t = g
=1
AND
element-wise multiplication
truth table = AND
σ
σ
tanh
σ
A h_{t-1}
h_t
X_t
B
OR
w x + w'h + b
40·0 + 40·0 - 20 = -20 → σ(-20) ~ 0
40·1 + 40·0 - 20 = 20 → σ(20) ~ 1
40·0 + 40·1 - 20 = 20 → σ(20) ~ 1
40·1 + 40·1 - 20 = 60 → σ(60) ~ 1
Showing OR operation

Figure 1: The flow with weights and biases of the Network for Parity

2 Learning to Copy Finite State Machines

2.1 Parity: Examining Generalization To Longer Sequences

TASK 2.1

In the implementation of a forward function for LSTM model for the parity task, there are 2 inputs for the forward function which is x(a padded input) and s (list of sequent lengths). The forward function contains 3 parts: a `pack_padded_sequence` function, a standard LSTM, a linear layer.

The input that we get as a parameter input to the forward function has a different dimension from what the `pack_padded_sequence` function requires, so we have to `unsqueeze` the input x to be able to parse to the `pack_padded_sequence` function without an error.

After the input dimension was updated, I then call `pack_padded_sequence` to optimize the computations, so we don't waste time and resources to compute unnecessary value. Then the output of this function will be the input of a standard `LSTM` function. I am using a built-in PyTorch LSTM function with the setting: `batch_first = True, input_size = 1, hidden_size = hidden_dim, num_layers = 1`. One of the output from this function is the hidden state, which I use the last index of the hidden state to the linear layer. The `linear` function has the setting: `in_features = hidden_dim, out_features = 2`. The output of the `linear` function is the output of this `forward` function.

We can see that I did not use the unpacking function for this task. The parity task is a sentence-level classification, so we do not care about the immediate hidden states.

With a standard LSTM and a linear layer, the program yields the result as below. We can see that the train accuracy reaches to 100% since around 160 epochs.

```
2021-02-01 08:25:53 INFO     epoch 130 train loss 0.393, train acc 0.855
2021-02-01 08:25:53 INFO     epoch 140 train loss 0.263, train acc 0.935
2021-02-01 08:25:53 INFO     epoch 150 train loss 0.150, train acc 0.984
2021-02-01 08:25:53 INFO     epoch 160 train loss 0.078, train acc 1.000
2021-02-01 08:25:53 INFO     epoch 170 train loss 0.040, train acc 1.000
2021-02-01 08:25:53 INFO     epoch 180 train loss 0.022, train acc 1.000
2021-02-01 08:25:53 INFO     epoch 190 train loss 0.014, train acc 1.000
2021-02-01 08:25:53 INFO     epoch 200 train loss 0.010, train acc 1.000
2021-02-01 08:25:53 INFO     epoch 210 train loss 0.008, train acc 1.000
```

TASK 2.2

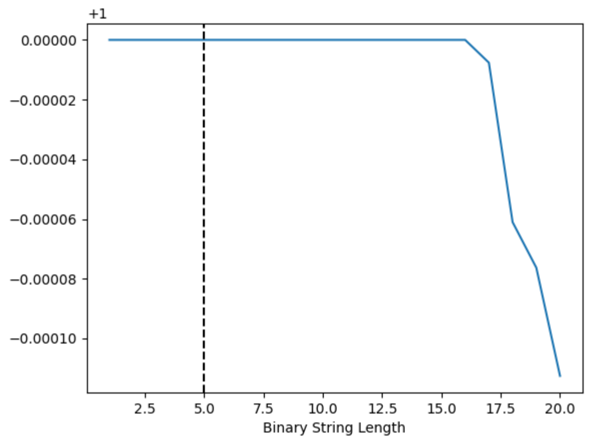


Figure 2: The validation accuracy rate when hidden_dim = 64, epochs = 2000, rl = 0.003

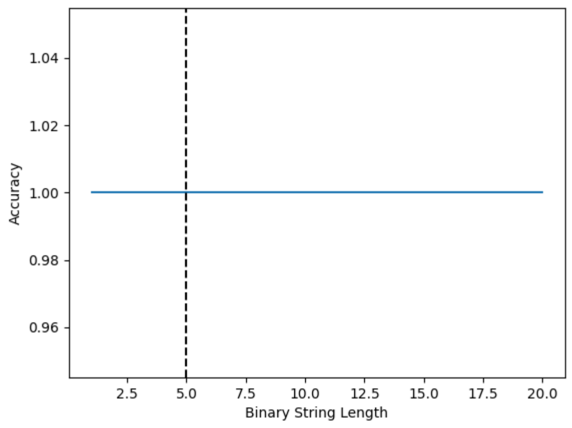


Figure 3: The validation accuracy rate when hidden_dim = 32, epochs = 2000, rl = 0.003

We trained the model on `max_length` equals to 5. The performance on the validation set is also very good when the the `max_length` is around 5. When the length comes to more than 15, the model seems to loss the generalization (overfit problem). When I reduce the hidden state size to 32, the validation accuracy rate is 100% until binary sequence of length 20.

TASK 2.3

From Figure 4-9, we can see that when the hidden dimension is smaller, the learning rate has to be higher to be able to maintain the high validation accuracy. It is easy to compare in the Figure 7 and 8, where the hidden state dimension is 2 with the epochs equal to 2000 in the different learning rate. The learning rate 0.05 yields the better validation accuracy rate than 0.03. I can reduce the hidden state dimension to 1 (The smallest size I can get) but the learning rate will be 0.8. When I reduce dimension, the model tends to learn slower, which would be a reason why we need to increase the learning rate.

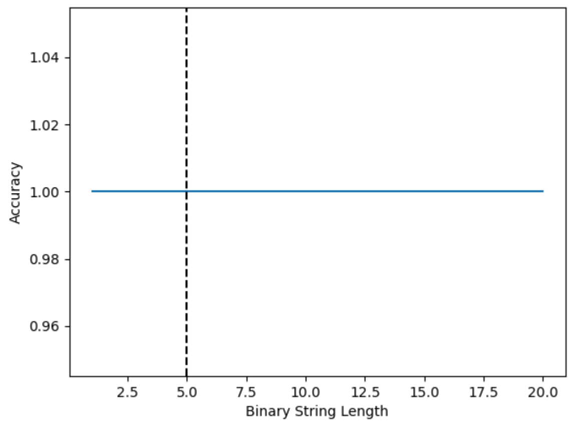


Figure 4: The validation accuracy rate when hidden_dim = 200, epochs = 2000, rl = 0.003

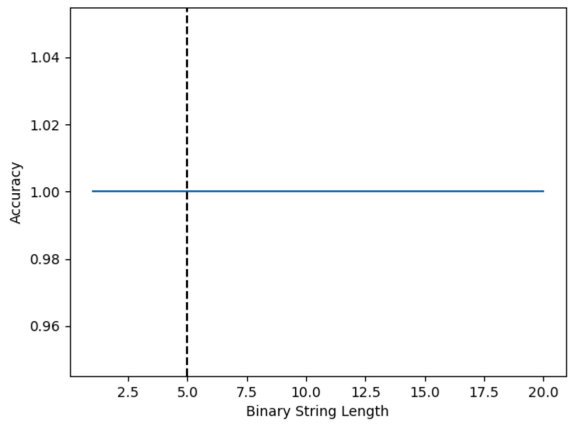


Figure 5: The validation accuracy rate when hidden_dim = 80, epochs = 2000, rl = 0.003

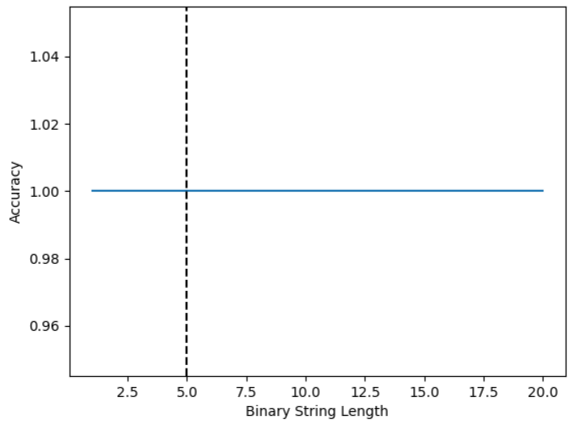


Figure 6: The validation accuracy rate when hidden_dim = 10, epochs = 2000, rl = 0.03

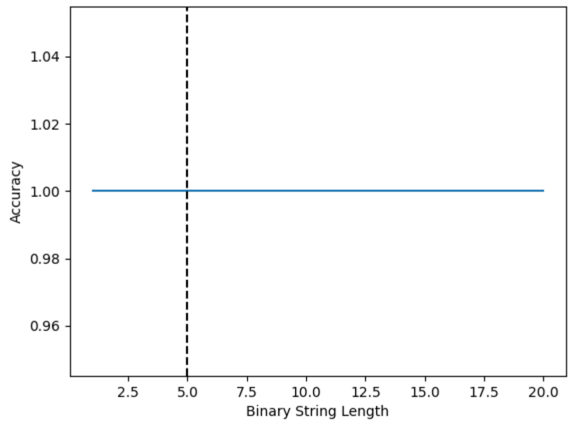


Figure 7: The validation accuracy rate when hidden_dim = 5, epochs = 2000, rl = 0.03

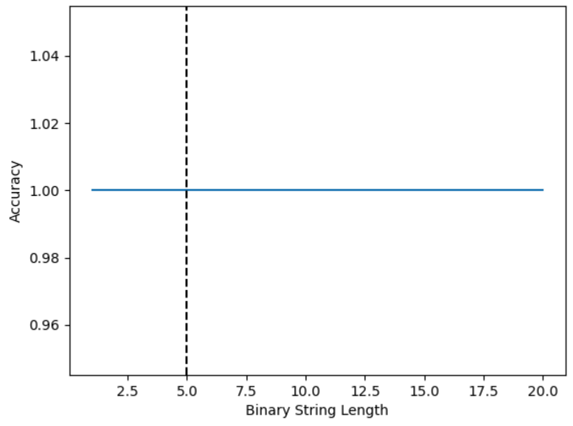


Figure 8: The validation accuracy rate when hidden_dim = 3, epochs = 2000, rl = 0.03

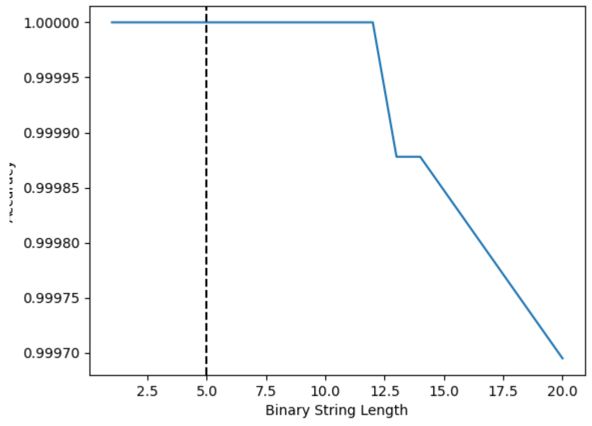


Figure 9: The validation accuracy rate when hidden_dim = 2, epochs = 2000, rl = 0.03

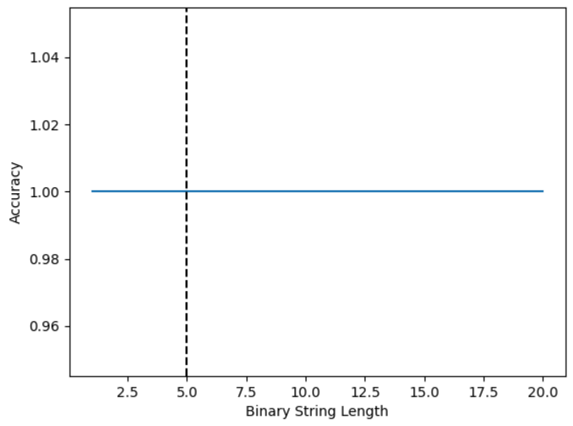


Figure 10: The validation accuracy rate when hidden_dim = 2, epochs = 2000, rl = 0.05

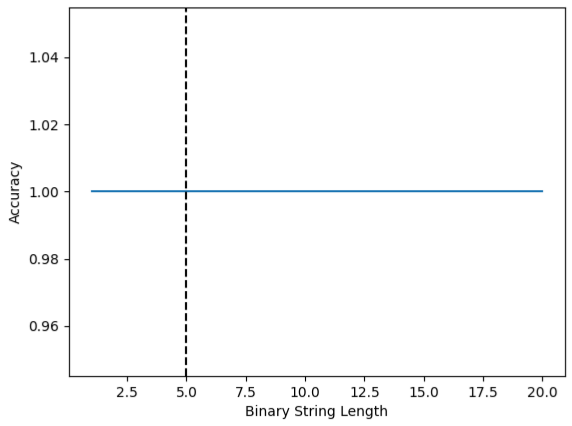


Figure 11: The validation accuracy rate when hidden_dim = 1, epochs = 2000, rl = 0.8

TASK 2.4

Vanilla RNNs have a serious problem in vanish gradient. The cause of this problem is vanilla RNNs calculate the gradient through all layers and through time. When vanilla RNNs was trained on the long distance and the time past, they could not use some of the earlier information. On the other hand, LSTMs has the cell states can be used as a bypass through the network. This is the reason that LSTMs has the capability to remember the longer sequences than vanilla RNNs.

When we evaluate the validation accuracy, the vanilla RNN couldn't remember the earlier state of the model so it seems to train base on the later information. As the model tries to fit the training set, this is why it impact only the validation set.

3 Part-of-Speech Tagging

TASK 3.1

The first 1,000 sentences' topic is related to the politic. For example:

pakistan now has a large force deployed in baluchistan which was not there before.
afghanistan has had the benefit of cooperation from both the us and iran
he clearly enjoyed , as governor , watching executions
this year a newspaper claimed two uk firms were involved in a deal in which thousands of guns for iraqi forces were re-routed to al - qaeda

For the 2,000th-3,000th sentence, the format is the email format. For example:

mary.ellenberger@enron.com on 05/03/2001 04:06:52 pm please respond to mary.ellenberger@enron.com cc : subject : re : if tgpl la z1 jan 9.95 feb 6.25 mar 4.98 april 5.37 esimien@nisource.com on 05/03/2001 01:32:35 pm hi mary , please do me a favour and give me the subject price for jan , feb , mar and apr 2001 . thanks , gregg penman
note : the information in this email is confidential and may be legally privileged .

For the 9,000th-10,000th sentence, They are reviews. For example:

because of the ants i dropped them to a 3 star .
cj and company did all that we ask and 10 times more .
excellent customer service and honest feedback .

These information, that stay in groups by the order of examples, will give us an idea why we should shuffle the data. If we do not shuffle the data, the model will learn group by group and eventually the model will have bad performance.

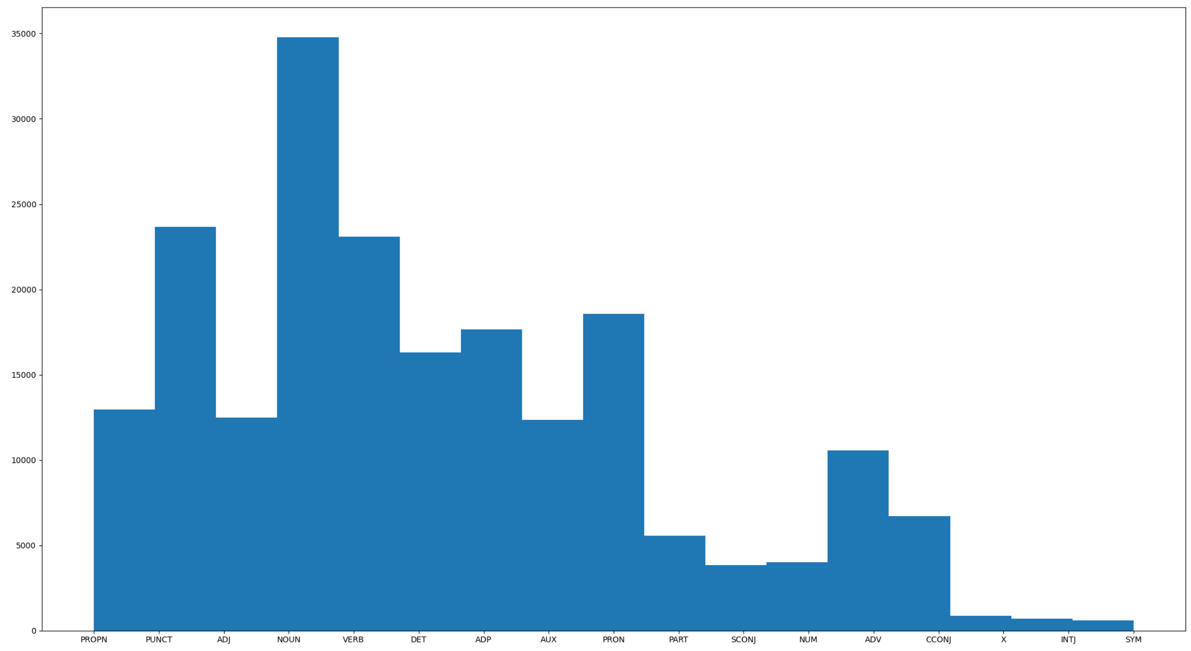


Figure 12: The histogram of part-of-speech tags

```
Tag                 Count               Percentage

NOUN                34781               17.00 %
PUNCT               23679               11.57 %
VERB                23081               11.28 %
PRON                18577                9.08 %
ADP                 17638                8.62 %
DET                 16285                7.96 %
PROPN               12946                6.33 %
ADJ                 12477                6.10 %
AUX                 12343                6.03 %
ADV                 10548                5.16 %
CCONJ               6707                 3.28 %
PART                5567                 2.72 %
NUM                 3999                 1.95 %
SCONJ               3843                 1.88 %
X                   847                  0.41 %
INTJ                688                  0.34 %
SYM                 599                  0.29 %
```

We can see from the histogram in Figure 12 and the result of counting number of tags that the data is not balance. The top 3 highest amount of tags are noun, verb, and punctuation, while interjection, symbol, and other types are 3 smallest in size.

Lemmatization and stemming are not appropriate approaches for this dataset. They will make the model even worse. For example, lemmatization could change the adjective form to noun form, and the model will lose the information of predicting either noun or adjective. Stemming will cut the end part of the word, which we will lost the information of part-of-speech tagging. We could normalize but it is not necessary. For example, when we normalize from n't to not, after model learn, they might give n't or not to the same tag.

It is obvious that text preprocessing depends on both dataset and task/problem.

For this dataset, the tag that has highest percentage is NOUN, which is 17%. If the model always return NOUN for every word, we will have the word-level accuracy to be 17%.

TASK 3.2

The train/validation/test accuracies from `driver_udpos.py` are shown as below. The test accuracy rate is around 89.356%.

```
epoch 1    train loss 1.541, train acc 57.305%     val loss 0.902, val acc 77.084%
epoch 2    train loss 0.450, train acc 88.069%     val loss 0.554, val acc 85.500%
epoch 3    train loss 0.268, train acc 92.611%     val loss 0.465, val acc 87.517%
epoch 4    train loss 0.202, train acc 94.294%     val loss 0.427, val acc 88.315%
epoch 5    train loss 0.164, train acc 95.245%     val loss 0.407, val acc 89.040%
epoch 6    train loss 0.137, train acc 96.055%     val loss 0.392, val acc 89.341%
epoch 7    train loss 0.117, train acc 96.621%     val loss 0.384, val acc 89.499%
epoch 8    train loss 0.101, train acc 97.105%     val loss 0.380, val acc 89.609%
epoch 9    train loss 0.088, train acc 97.505%     val loss 0.382, val acc 89.803%
epoch 10   train loss 0.077, train acc 97.818%     val loss 0.380, val acc 89.892%

test loss 0.392, test acc 89.356%
```

Figure 13 shows training and validation loss for 10 epochs. In this model I used pretrained token embedding (GloVe) as from the suggestion in the assignment description, which help the model has better performance. I have tried 100, 200, 300 dimensions of this embedding. They all yielded the similar results. I use Adam as an optimizer. The learning rate that I had tried was 0.01, 0.001, and 0.0001. The best validation accuracy is when learning rate = 0.001. The next hyper-parameter that I had tried was weight_decay. The model yielded very similar result for the values 0.00001, 0.0000001, or

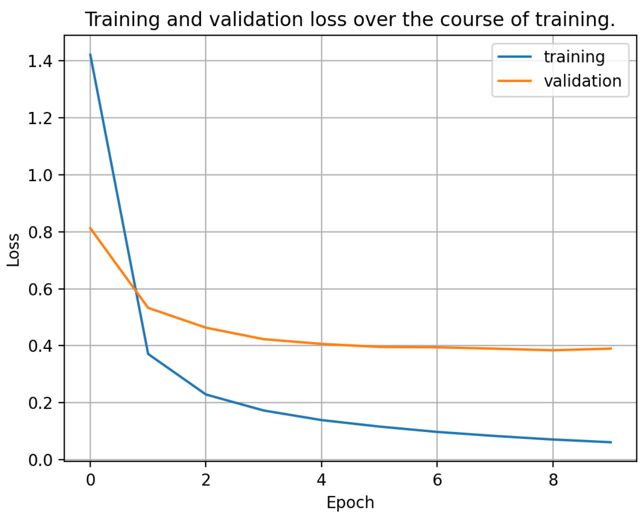


Figure 13: Training and validation loss

0. The hidden dimension was the next things that I experimented. The large hidden dimension (64, 128: similar results) gave the higher accuracy rate while the small dimension (10) gave the lower accuracy rate. I tried to run many size of epoch (10, 20, 30). I could not see much different from these 3 values, so I chose the smallest one.

I also kept track of the lowest validation loss model to use in the prediction of test data. This is the early-stopping technique that I used for this work.

The above result and graph's parameter is lr = 0.001, weight decay = 0.000001, epoch = 10, hidden dimension = 128, embedding dimension = 100, batch size = 128

TASK 3.3

The results from `tag_sentence` function is shown as below.

```
TAG                 TOKEN

DET                 the
ADJ                 old
NOUN                man
DET                 the
NOUN                boat
PUNCT               .
------------------------------
TAG                 TOKEN

PROPN               The
ADJ                 complex
NOUN                houses
VERB                married
CCONJ               and
ADJ                 single
NOUN                soldiers
CCONJ               and
PRON                their
NOUN                families
PUNCT               .
------------------------------
```

```
TAG                 TOKEN

PROPN               The
NOUN                man
PRON                who
VERB                hunts
VERB                ducks
ADP                 out
ADP                 on
NOUN                weekends
PUNCT               .
```