Homework 4 Final Report

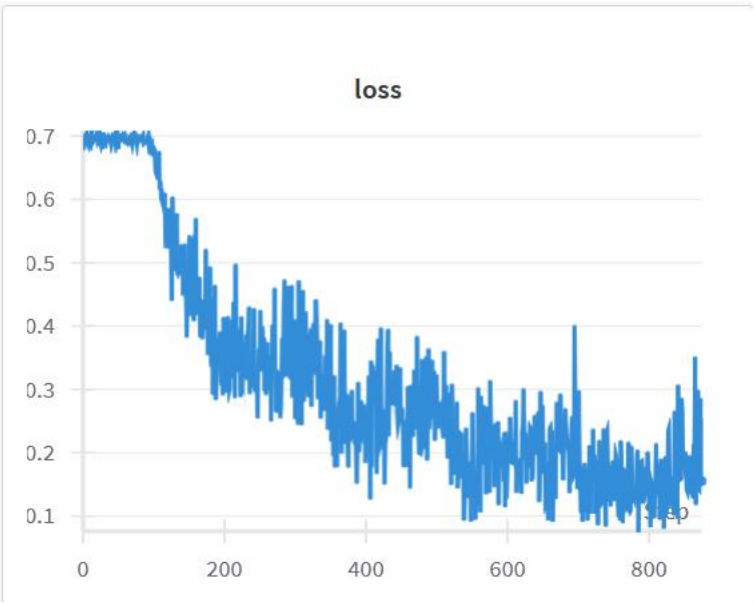
Arden Walker

Test Accuracy and Other Experimental Results:

The test accuracy when embedding_dim was set to 100 was 90%. Hyper-parameters were as follows. Two LSTM layers were chosen. The batch size was set to 120 to allow for better training. The hidden dimension was increased to 200 for a similar reason. The clip was decreased to 3. The number of epoches was 5 and the learning rate was 0.001.

```
## ------------------------------------------------------
## please change the parameter settings by yourselves
## ------------------------------------------------------
mode = 'train'
Batch_size =120
n_layers = 2 ## choose 1-3 layers

## input seq length aligned with data pre-processing
input_len = 150

## word embedding length
embedding_dim = 100

# lstm hidden dim
hidden_dim = 200
# binary cross entropy
output_size = 1
num_epoches = 5
## please change the learning rate by youself
learning_rate = 0.001
# gradient clipping
clip = 3
load_cpt = False # True
ckp_path = 'cpt/name.pt'
# embedding_matrix = None
## use pre-train Glove embedding or not?
pretrain = False #True
```

```
**** save checkpoint ****
----model testing now----
wandb: Currently logged in as: akwalker (akwalker-william-mary) to https://api.wandb.ai. Use `wandb login --relogin` to
force relogin
wandb: Tracking run with wandb version 0.19.8
wandb: Run data is saved locally in C:\Users\ardw1\Data_Mining\Assignment-4\HW4-code-template\wandb\run-20250506_200457-
oackyxai
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run misty-sea-68
wandb:  View project at https://wandb.ai/akwalker-william-mary/HW4
wandb:  View run at https://wandb.ai/akwalker-william-mary/HW4/runs/oackyxai
C:\Users\ardw1\Data_Mining\Assignment-4\HW4-code-template\DataLoader.py:29: UserWarning: To copy construct from a tensor
, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather t
han torch.tensor(sourceTensor).
  return torch.tensor(input_x), torch.tensor(label,dtype=torch.float)
Test accuracy:
90.0
Confusion matrix:
[[22  2]
 [ 2 14]]
running time:  8.586571852366129 mins
```



Based on the above confusion matrix, True Positive = 22, False Negative = 2, False Positive = 2, and True Negative = 14:

$$\textit{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Negative + False\ Positive + True\ Negative} = \frac{22+14}{22+2+2+14} = 0.90$$

$$\textit{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{22}{22+2} = 0.92$$

$$\textit{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{22}{22+2} = 0.92$$

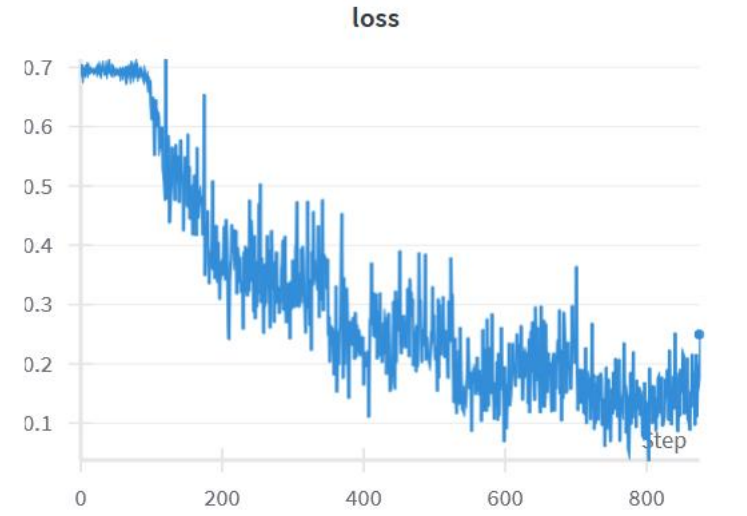
$$\textit{F1 Score} = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0.92 * 0.92}{0.92 + 0.92} = 0.92$$

When the embedding dimension was set to 200, the test accuracy was 85%. Hyper-parameters were as follows. They were all identical to embedding dimension of 100

hyperparameters, except the embedding dimension was increased to 200 and the hidden dimension decreased to 150.

```
## ------------------------------------------------
## please change the parameter settings by yourselves
## ------------------------------------------------
mode = 'train'
Batch_size =120
n_layers = 2 ## choose 1-3 layers

## input seq length aligned with data pre-processing
input_len = 150

## word embedding length
embedding_dim = 200

# lstm hidden dim
hidden_dim = 150
# binary cross entropy
output_size = 1
num_epoches = 5
## please change the learning rate by youself
learning_rate = 0.001
# gradient clipping
clip = 3
load_cpt = False # True
ckp_path = 'cpt/name.pt'
# embedding_matrix = None
## use pre-train Glove embedding or not?
pretrain = False #True
```

```
**** save checkpoint ****
----model testing now----
wandb: Currently logged in as: akwalker (akwalker-william-mary) to https://api.wandb.ai. Use `wandb login --relogin` to
force relogin
wandb: Tracking run with wandb version 0.19.8
wandb: Run data is saved locally in C:\Users\ardw1\Data_Mining\Assignment-4\HW4-code-template\wandb\run-20250506_194637-
6pd1zrzl
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run peach-jazz-61
wandb:  View project at https://wandb.ai/akwalker-william-mary/HW4
wandb:  View run at https://wandb.ai/akwalker-william-mary/HW4/runs/6pd1zrzl
C:\Users\ardw1\Data_Mining\Assignment-4\HW4-code-template\DataLoader.py:29: UserWarning: To copy construct from a tensor
, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather t
han torch.tensor(sourceTensor).
  return torch.tensor(input_x), torch.tensor(label,dtype=torch.float)
Test accuracy:
85.0
Confusion matrix:
[[19  5]
 [ 1 15]]
running time:  5.341720124085744 mins
```



From the confusion matrix, True Positive = 19, False Negative = 5, False Positive = 1, and True Negative = 15:

$$\textit{Accuracy} = \frac{True\ Positive+True\ Negative}{True\ Positive+False\ Negative+False\ Positive+True\ Negative} = \frac{19+15}{19+5+1+15} = 0.85$$

$$\textit{Precision} = \frac{True\ Positive}{True\ Positive+False\ Positive} = \frac{19}{19+1} = 0.95$$

$$\textit{Recall} = \frac{True\ Positive}{True\ Positive+False\ Negative} = \frac{19}{19+5} = 0.80$$

$$\textit{F1 Score} = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*0.95*0.80}{0.95+0.80} = 0.86$$

Model Architecture:

First, complete data preprocessing. Load in the training and test .csv files:

```
'''step 1: read the raw data'''
def _read_data():
    ##-----------------------------------------
    ## read the training and test data from files
    ## *** complete the code below ***
    ##-----------------------------------------
    train_data = pd.read_csv('training_raw_data.csv',index_col=None,encoding='utf8')
    test_data = pd.read_csv('test_raw_data.csv',index_col=None,encoding='utf8')

    return train_data, test_data
```

Second, convert the words to lowercase by applying the lambda function.

```
##-----------------------------------------------------
## complete code below to convert words to lower case
##-----------------------------------------------------
input_data['clean_text'] = input_data['clean_text'].apply(lambda x: x.lower())
```

Now, convert the “pos” and “neg” labels for each review to “1” and “0,” respectively.

```
##-------------------------------------------------
## complete code below to convert "pos" and "neg"
## to boolean values 1 and 0
##-------------------------------------------------
input_data['Label'] = input_data['Label'].apply(lambda x: 0 if x=='neg' else 1)
```

The next step is to convert unknown to the index “1” and padding to the index “0”.

```
""
## complete code to add index for padding (0) and unknown (1)
##-------------------------------------------------------------
tokens2index['<pad>'] = 0
tokens2index['<unk>'] = 1
```

Finally, add a padding (0) to the list of words that is too short. Dynamically add zeroes depending on how much shorter than optimal the given list is.

```
##-------------------------------------------------------------
## complete code to add padding "0" to the end of list of words
##-------------------------------------------------------------
return x+[0]*(seq_len-len(x))
```

Now that the preprocessing is complete, finish the code for the data loader. Convert the labels and features using torch.tensor():

```
##-----------------------------------------------
## complete the code to load features and labels
##-----------------------------------------------
input_x = torch.tensor(input_x)
label = torch.tensor(label,dtype=torch.float)
```

First, feed in the words to get their appropriate embeddings.

```
##-----------------------------------------------------
## complete code to feed input sequence x to get embeddings
##-----------------------------------------------------
embeds = self.embedding(x)
```

Now, feed in the input layer and output the hidden layer:

```
##-------------------------------------------------------
## complete code to feed input sequence x to get embeddings
##-------------------------------------------------------
lstm out.   = self.lstm(embeds. hidden cell)
```

Finally, the output layer must be pooled to reduce the sequence length dimension. Do this using self.pool() defined above previously (see LSTM.py).

```
##-------------------------------------------------------
## complete code to pool the output to reduce seq_len dim
##-------------------------------------------------------
out = self.pool(lstm_out)
```

The code in main must now be completed. Import the LSTM model with predefined parameters such as vocab_size, output_size, and so on. Load the model to the device.

```
## ---------------------------------------------
## step 4: import model from LSTM.py
## complete the code in "def forward(self, x)" in LSTM.py file
## then import model from LSTM.py below
## and also load model to device
## ---------------------------------------------
model = LSTMModel(vocab_size, output_size, embedding_dim, embedding_matrix, hidden_dim, n_layers, input_len)
model.to(device)
```

Use the Adam optimizer and Binary Cross Entropy Loss.

```
##-------------------------------------------------------------
## step 5: complete code to define optimizer and loss function
##-------------------------------------------------------------
optimizer = optim.Adam(model.parameters(),lr=learning_rate)  #
## define Binary Cross Entropy Loss below
loss_fun = nn.BCELoss()
```

Load the checkpoint called “ckp_path.” Do this every epoch.

```
##-----------------------------------------------
## complete code below to load checkpoint
##-----------------------------------------------
checkpoint = torch.load(ckp_path)
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoches = checkpoint['epoch']
```

Obtain the prediction result from the model with y_out. Send the batch of data through the model to achieve this.

```
##-----------------------------------------------
## complete code to get predict result from mode
##-----------------------------------------------
y_out = model(x_batch)
```

Send the expected results (y_labels) and predicted results (y_out) through the loss function.

```
##-----------------------------------------------
## complete code to get loss
##-----------------------------------------------
loss = loss_fun(y_out, y_labels)
```

Save the checkpoint. A variable global_step was previously defined and initialized to 0, and it is updated after each iteration.

```
##-----------------------------------------------
## step 9: complete code below to save checkpoint
##-----------------------------------------------
print("**** save checkpoint ****")
ckp_path = 'checkpoint/step_{}.pt'.format(global_step)
_save_checkpoint(ckp_path, model, epoches, global_step, optimizer)
global_step = global_step + 1 # go to the next training iteration
```

Lastly, do model testing by specifying no gradient. Get the prediction result from passing the batch data through the model, and round the results from a decimal to 0 or 1. Compute the test accuracy and display the result. Also, import confusion matrix and use it to calculate the matrix accuracy, precision, recall, and F1 Score (see this calculation in **Experimental Results**).

```
##-----------------------------------------------------------------
## step 10: complete code below for model testing
## predict result is a single value between 0 and 1, such as 0.8, so
## we can use y_pred = torch.round(y_out) to predict label 1 or 0
##-----------------------------------------------------------------
print("----model testing now----")
model.eval()
with torch.no_grad():
        for x_batch, y_labels in test_generator:
            x_batch, y_labels = x_batch.to(device), y_labels.to(device)
            y_out = model(x_batch)
            y_pred = torch.round(y_out)

#NEW
accy = 0
for image in range(y_labels.size(0)):
    if y_pred[image] == y_labels[image]:
            accy += 1 # count the image as correctly classified.
accy = accy / y_labels.size(0) * 100
print("Test accuracy:")
print(accy)

matrix = confusion_matrix(y_labels, y_pred)
print("Confusion matrix:")
print(matrix)
```

Source Code:

Please see the completed code in HW4-code-template. Test_data.csv and training_data.csv are the output from this project, converting the labels from positive and negative to 1 and 0. Below is first an image of test_data.csv, and then an image of training_data.csv.

Content	Label	seq_len	clean_text	seq_words	inpu
How truly f	1	294	how truly fi	['how', 'trul	[88,
What a ter	0	171	what a terr	['what', 'a',	[49,
Stephen Ki	0	498	stephen ki	['stephen',	[160
Firstly, I wo	0	328	firstly i wo	['firstly', 'i',	[419
This show	0	230	this show i	['this', 'sho	[12,
Ulises is a	0	145	ulises is a	['ulises', 'is	[1, 7
Great mov	1	132	great movi	['great', 'm	[78,

Content	Label	seq_len	clean_text	seq_words	input_x
As much a	0	237	as much a	['as', 'mucl	[17, 75, 17,
"The polar	0	370	the polar e	['the', 'pola	[2, 7574, 26
The British	1	190	the british	['the', 'briti	[2, 680, 335
The first fil	0	240	the first fil	['the', 'first	[2, 84, 21, 7
This show	0	102	this show c	['this', 'sho	[12, 113, 57
NOBODY (	1	181	nobody 19	['nobody', '	[1237, 4148
Three word	1	236	three word	['three', 'w	[294, 724, 4

"Bonus Section"

Load the pre-train embedding for dim = 200 and dim = 300.

First, do dim = 200. Ensure embedding_dim is set to 200 and update the following:

```
## word embedding length
embedding_dim = 200
```

```
## use pre-train Glove embedding or not?
pretrain = True #False

##--------------------------------------------------------------------
## Bonus (5%): complete code to add GloVe embedding file path below.
## Download Glove embedding from https://nlp.stanford.edu/data/glove.6B.zip
## "embedding_dim" defined above shoud be aligned with the dimension of GloVe embedddings
## if you do not want bonus, you can skip it.
##--------------------------------------------------------------------
glove_file = 'C:/Users/ardw1/Data_Mining/Assignment-4/Hw4-code-template/glove.6B.200d.txt' ## change by yourself
```

Now, the glove embedding is loaded before the model training begins.

```
***** load glove embedding now...****
********************************************************************************
start model training now
********************************************************************************
```

The test accuracy is below. (87.5%)

```
wandb: Currently logged in as: akwalker (akwalker-william-mary) to https://api.wandb.ai. Use `wandb login --relogin` t
force relogin
wandb: Tracking run with wandb version 0.19.8
wandb: Run data is saved locally in C:\Users\ardw1\Data_Mining\Assignment-4\HW4-code-template\wandb\run-20250506_1938
xg7tqhci
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run solar-butterfly-54
wandb:  View project at https://wandb.ai/akwalker-william-mary/HW4
wandb:  View run at https://wandb.ai/akwalker-william-mary/HW4/runs/xg7tqhci
C:\Users\ardw1\Data_Mining\Assignment-4\HW4-code-template\DataLoader.py:29: UserWarning: To copy construct from a ten
, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rathe
han torch.tensor(sourceTensor).
  return torch.tensor(input_x), torch.tensor(label,dtype=torch.float)
Test accuracy:
87.5
```

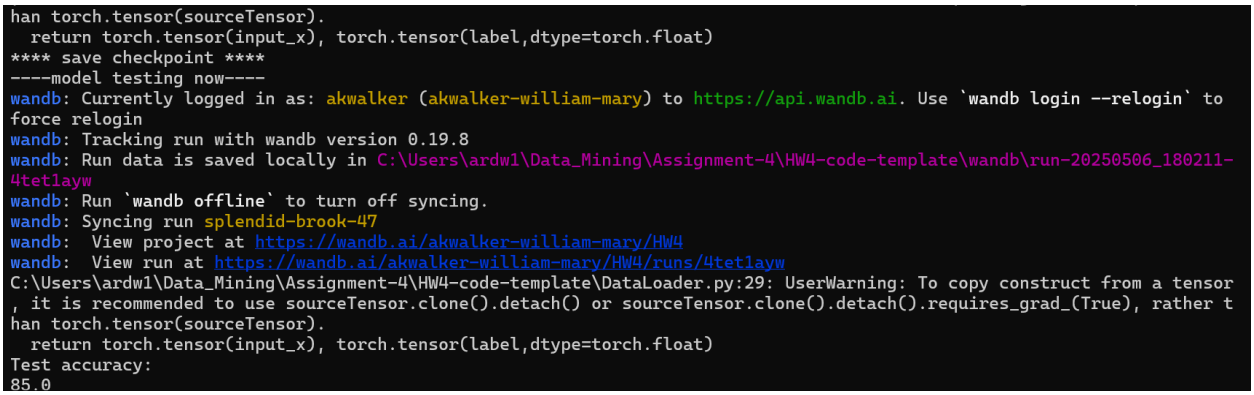
Now, run with dim = 300.

```
## please change the parameter settings by yourselves
## ------------------------------------------------------
mode = 'train'
Batch_size =120
n_layers = 2 ## choose 1-3 layers

## input seq length aligned with data pre-processing
input_len = 150

## word embedding length
embedding_dim = 300

# lstm hidden dim
hidden_dim = 200
# binary cross entropy
output_size = 1
num_epoches = 5
## please change the learning rate by youself
learning_rate = 0.001
# gradient clipping
clip = 3
load_cpt = False # True
ckp_path = 'cpt/name.pt'
# embedding_matrix = None
## use pre-train Glove embedding or not?
pretrain = True #False
```

Open glove.6b300d rather than 200d:

```
glove_file = 'C:/Users/ardw1/Data_Mining/Assignment-4/Hw4-code-template/glove.6B.300d.txt' ## change by yourself
```

This is the result (accuracy is 85%):



The test accuracy with pre-train embedding of 300 is the same as the test accuracy with the regular embedding set at 200 and lower than the test accuracy with the regular embedding set at 100. The test accuracy with pre-train embedding of 200 is between the accuracies of regular embeddings of 200 and 300 (87.5 is between 85 and 90).