# ECE695DL: Homework 7

Zeyu Zhou

26 April 2021

**hw07_task1.py**

```
# ECE 695DL 2021SPIRNG
# Homework7
# Zeyu Zhou



import random
import numpy
import torch
import torch.optim as optim
import copy
import math


import os, sys
# I did not install DLStudio. I directly import from local
↪  python files. To run it on your device, comment out or
↪  modify this line.
sys.path.append('F:\ECE 695 Deep
↪  Learning\Homework\HW7\DLStudio-2.0.8\DLStudio')

seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)
```

```
from DLStudio import *


class MySentimentAnalysisDataset(DLStudio.TextClassification.Sen⌋
↪  timentAnalysisDataset):

    # modified from corresponding code in DLStudio by Prof.Kak

    def __init__(self, train_or_test, dl_studio, dataset_file):
        super(DLStudio.TextClassification.SentimentAnalysisDatas⌋
        ↪  et,
        ↪  self).__init__()
        self.train_or_test = train_or_test
        root_dir = dl_studio.dataroot
        f = gzip.open(root_dir + dataset_file, 'rb')
        dataset = f.read()

        if train_or_test == 'train':
            if sys.version_info[0] == 3:
                self.positive_reviews_train,
                ↪  self.negative_reviews_train, self.vocab =
                ↪  pickle.loads(dataset, encoding='latin1')
            else:
                self.positive_reviews_train,
                ↪  self.negative_reviews_train, self.vocab =
                ↪  pickle.loads(dataset)

            self.categories =
            ↪  sorted(list(self.positive_reviews_train.keys()))
            self.category_sizes_train_pos = {category :
            ↪  len(self.positive_reviews_train[category]) for
            ↪  category in self.categories}
            self.category_sizes_train_neg = {category :
            ↪  len(self.negative_reviews_train[category]) for
            ↪  category in self.categories}


            ##################################################⌋
            ↪  ##################################
            ################################## encode into
            ↪  integers ################################
            ##################################################⌋
            ↪  ##################################
            self.word2id = dict()
            i = 0
```

```
    for word in self.vocab:
        self.word2id[word] = i/len(self.vocab)
        i += 1
    self.id2word = {i: word for word, i in
    ↪  self.word2id.items()}


    self.indexed_dataset_train = []
    for category in self.positive_reviews_train:
        for review in
        ↪  self.positive_reviews_train[category]:
            self.indexed_dataset_train.append([review,
            ↪  category, 1])
    for category in self.negative_reviews_train:
        for review in
        ↪  self.negative_reviews_train[category]:
            self.indexed_dataset_train.append([review,
            ↪  category, 0])
    random.shuffle(self.indexed_dataset_train)
elif train_or_test == 'test':
    if sys.version_info[0] == 3:
        self.positive_reviews_test,
        ↪  self.negative_reviews_test, self.vocab =
        ↪  pickle.loads(dataset, encoding='latin1')
    else:
        self.positive_reviews_test,
        ↪  self.negative_reviews_test, self.vocab =
        ↪  pickle.loads(dataset)
    self.vocab = sorted(self.vocab)
    self.categories =
    ↪  sorted(list(self.positive_reviews_test.keys()))
    self.category_sizes_test_pos = {category :
    ↪  len(self.positive_reviews_test[category]) for
    ↪  category in self.categories}
    self.category_sizes_test_neg = {category :
    ↪  len(self.negative_reviews_test[category]) for
    ↪  category in self.categories}

    ##################################################
    ↪  ################################
    ################################ encode into
    ↪  integers ##############################
    ##################################################
    ↪  ################################
    self.word2id = dict()
    i = 0
```

```
            for word in self.vocab:
                self.word2id[word] = i/len(self.vocab)
                i += 1
            self.id2word = {i: word for word, i in
            ↪  self.word2id.items()}

            self.indexed_dataset_test = []
            for category in self.positive_reviews_test:
                for review in
                ↪  self.positive_reviews_test[category]:
                    self.indexed_dataset_test.append([review,
                    ↪  category, 1])
            for category in self.negative_reviews_test:
                for review in
                ↪  self.negative_reviews_test[category]:
                    self.indexed_dataset_test.append([review,
                    ↪  category, 0])
            random.shuffle(self.indexed_dataset_test)


    def review_to_integer_tensor(self, review):
        review_tensor = torch.zeros(len(review), 1)
        for i,word in enumerate(review):
            review_tensor[i,:] = self.word2id[word]
        return review_tensor

    def __getitem__(self, idx):
        sample = self.indexed_dataset_train[idx] if
        ↪  self.train_or_test == 'train' else
        ↪  self.indexed_dataset_test[idx]
        review = sample[0]
        review_category = sample[1]
        review_sentiment = sample[2]
        review_sentiment =
        ↪  self.sentiment_to_tensor(review_sentiment)
        review_tensor = self.review_to_integer_tensor(review)
        category_index = self.categories.index(review_category)
        sample = {'review'       : review_tensor,
                  'category'     : category_index, # should be
                  ↪  converted to tensor, but not yet used
                  'sentiment'    : review_sentiment }
        return sample


dls = DLStudio(   dataroot =
↪  "./data/TextDatasets/sentiment_dataset/",
```

```
                    path_saved_model = "./saved_model",
                    momentum = 0.9,
                    learning_rate =  1e-5,
                    epochs = 5 ,
                    batch_size = 1,
                    classes = ('negative','positive'),
                    use_gpu = True,
                )



text_cl = DLStudio.TextClassification( dl_studio = dls )

dataserver_train = MySentimentAnalysisDataset(
                                    train_or_test = 'train',
                                    dl_studio = dls,
                                    dataset_file = "sentiment_datas⌋
                                    ↪  et_train_40.tar.gz",
#                                   dataset_file =
↪  "sentiment_dataset_train_200.tar.gz",
                    )
dataserver_test = MySentimentAnalysisDataset(
                                    train_or_test = 'test',
                                    dl_studio = dls,
                                    dataset_file = "sentiment_datas⌋
                                    ↪  et_test_40.tar.gz",
#                                   dataset_file =
↪  "sentiment_dataset_train_200.tar.gz",
                    )
text_cl.dataserver_train = dataserver_train
text_cl.dataserver_test = dataserver_test

text_cl.load_SentimentAnalysisDataset(dataserver_train,
↪  dataserver_test)

vocab_size = dataserver_train.get_vocab_size()

model = text_cl.TEXTnetOrder2(1, hidden_size=512, output_size=2)

number_of_learnable_params = sum(p.numel() for p in
↪  model.parameters() if p.requires_grad)
num_layers = len(list(model.parameters()))

print("\n\nThe number of layers in the model: %d" % num_layers)
print("\nThe number of learnable parameters in the model: %d" %
↪  number_of_learnable_params)
```

```
text_cl.run_code_for_training_with_TEXTnetOrder2(model,
↪  hidden_size=512)


text_cl.run_code_for_testing_with_TEXTnetOrder2(model,
↪  hidden_size=512)
```

**hw07_task2.py**

```
# ECE 695DL 2021SPIRNG
# Homework7
# Zeyu Zhou



import random
import numpy
import torch
import torch.optim as optim
import torch.nn as nn
import copy
import math
from time import time
import gzip

import matplotlib.pyplot as plt
import os, sys

# I did not install DLStudio. I directly import from local
↪  python files. To run it on your device, comment out or
↪  modify this line.
sys.path.append('F:\ECE 695 Deep
↪  Learning\Homework\HW7\DLStudio-2.0.8\DLStudio')

seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)
```

```
##  watch -d -n 0.5 nvidia-smi

from DLStudio import *



class MySentimentAnalysisDataset(DLStudio.TextClassification.Sen⌋
↪  timentAnalysisDataset):

    # modified from corresponding code in DLStudio by Prof.Kak

    def __init__(self, train_or_test, dl_studio, dataset_file):
        super(DLStudio.TextClassification.SentimentAnalysisDatas⌋
        ↪  et,
        ↪  self).__init__()
        self.train_or_test = train_or_test
        root_dir = dl_studio.dataroot
        f = gzip.open(root_dir + dataset_file, 'rb')
        dataset = f.read()

        if train_or_test == 'train':
            if sys.version_info[0] == 3:
                self.positive_reviews_train,
                ↪  self.negative_reviews_train, self.vocab =
                ↪  pickle.loads(dataset, encoding='latin1')
            else:
                self.positive_reviews_train,
                ↪  self.negative_reviews_train, self.vocab =
                ↪  pickle.loads(dataset)

            self.categories =
            ↪  sorted(list(self.positive_reviews_train.keys()))
            self.category_sizes_train_pos = {category :
            ↪  len(self.positive_reviews_train[category]) for
            ↪  category in self.categories}
            self.category_sizes_train_neg = {category :
            ↪  len(self.negative_reviews_train[category]) for
            ↪  category in self.categories}


            ##################################################⌋
            ↪  ##################################
            ################################## encode into
            ↪  integers ################################
```

```
    ##################################################
    ################################
    self.word2id = dict()
    i = 0
    for word in self.vocab:
        #self.word2id[word] = i/len(self.vocab)
        self.word2id[word] = i
        i += 1
    self.id2word = {i: word for word, i in
    self.word2id.items()}


    self.indexed_dataset_train = []
    for category in self.positive_reviews_train:
        for review in
        self.positive_reviews_train[category]:
            self.indexed_dataset_train.append([review,
            category, 1])
    for category in self.negative_reviews_train:
        for review in
        self.negative_reviews_train[category]:
            self.indexed_dataset_train.append([review,
            category, 0])
    random.shuffle(self.indexed_dataset_train)
elif train_or_test == 'test':
    if sys.version_info[0] == 3:
        self.positive_reviews_test,
        self.negative_reviews_test, self.vocab =
        pickle.loads(dataset, encoding='latin1')
    else:
        self.positive_reviews_test,
        self.negative_reviews_test, self.vocab =
        pickle.loads(dataset)
    self.vocab = sorted(self.vocab)
    self.categories =
    sorted(list(self.positive_reviews_test.keys()))
    self.category_sizes_test_pos = {category :
    len(self.positive_reviews_test[category]) for
    category in self.categories}
    self.category_sizes_test_neg = {category :
    len(self.negative_reviews_test[category]) for
    category in self.categories}

    ##################################################
    ################################
```

```
        ################################## encode into
        ↪  integers ################################
        ##################################################
        ↪  ##################################
        self.word2id = dict()
        i = 0
        for word in self.vocab:
            #self.word2id[word] = i/len(self.vocab)
            self.word2id[word] = i
            i += 1
        self.id2word = {i: word for word, i in
        ↪  self.word2id.items()}

        self.indexed_dataset_test = []
        for category in self.positive_reviews_test:
            for review in
            ↪  self.positive_reviews_test[category]:
                self.indexed_dataset_test.append([review,
                ↪  category, 1])
        for category in self.negative_reviews_test:
            for review in
            ↪  self.negative_reviews_test[category]:
                self.indexed_dataset_test.append([review,
                ↪  category, 0])
        random.shuffle(self.indexed_dataset_test)


def review_to_integer_tensor(self, review):
    review_tensor = torch.zeros(len(review), 1)
    for i,word in enumerate(review):
        review_tensor[i,:] = self.word2id[word]
    return review_tensor

def __getitem__(self, idx):
    sample = self.indexed_dataset_train[idx] if
    ↪  self.train_or_test == 'train' else
    ↪  self.indexed_dataset_test[idx]
    review = sample[0]
    review_category = sample[1]
    review_sentiment = sample[2]
    review_sentiment =
    ↪  self.sentiment_to_tensor(review_sentiment)
    review_tensor = self.review_to_integer_tensor(review)
    category_index = self.categories.index(review_category)
    sample = {'review'       : review_tensor,
```

```
                    'category'     : category_index, # should be
                    ↪  converted to tensor, but not yet used
                    'sentiment'    : review_sentiment }
        return sample



class MyTextClassification(DLStudio.TextClassification):

    # modified from corresponding code in DLStudio by Prof.Kak

    def run_code_for_training_with_TEXTnetOrder2(self, net,
    ↪  hidden_size):
        filename_for_out = "performance_numbers_" +
        ↪  str(self.dl_studio.epochs) + ".txt"
        FILE = open(filename_for_out, 'w')
        net = copy.deepcopy(net)
        net = net.to(self.dl_studio.device)
        ## Note that the TEXTnet and TEXTnetOrder2 both produce
        ↪  LogSoftmax output:
        #criterion = nn.NLLLoss()
        #criterion = nn.BCEWithLogitsLoss()
        criterion = nn.MSELoss()
        #criterion = nn.BCELoss()
        accum_times = []
        optimizer = torch.optim.Adam(net.parameters(),
        ↪  lr=self.dl_studio.learning_rate, betas=(0.5, 0.999),
        ↪  eps=1e-08)
        #optimizer = optim.SGD(net.parameters(),
#                       lr=self.dl_studio.learning_rate,
↪  momentum=self.dl_studio.momentum)
        start_time = time.perf_counter()
        training_loss_tally = []
        net.train()
        for epoch in range(self.dl_studio.epochs):
            print("")
            running_loss = 0.0
            for i, data in enumerate(self.train_dataloader):
                cell_prev = net.initialize_cell(1).to(self.dl_st⌋
                ↪  udio.device)
                cell_prev_2_prev = net.initialize_cell(1).to(sel⌋
                ↪  f.dl_studio.device)
                hidden = torch.zeros(1, hidden_size)
                hidden = hidden.to(self.dl_studio.device)
```

```
            review_tensor,category,sentiment =
            ↪  data['review'], data['category'],
            ↪  data['sentiment']
            review_tensor =
            ↪  review_tensor.to(self.dl_studio.device)
            sentiment = sentiment.to(self.dl_studio.device)
            optimizer.zero_grad()
            input = torch.zeros(1,review_tensor.shape[2])
            input = input.to(self.dl_studio.device)
            for k in range(review_tensor.shape[1]):
                input[0,:] = review_tensor[0,k]
                output, hidden, cell = net(input, hidden,
                ↪  cell_prev_2_prev)
                if k == 0:
                    cell_prev = cell
                else:
                    cell_prev_2_prev = cell_prev
                    cell_prev = cell

            #loss = criterion(output,
            ↪  torch.argmax(sentiment,1))
            loss =
            ↪  criterion(output.float(),sentiment.float())

            running_loss += loss.item()
            loss.backward()
            optimizer.step()
            if i % 500 == 499:
                avg_loss = running_loss / float(500)
                training_loss_tally.append(avg_loss)
                current_time = time.perf_counter()
                time_elapsed = current_time-start_time
                print("[epoch:%d  iter:%4d  elapsed_time:
                ↪  %4d secs]     loss: %.5f" %
                ↪  (epoch+1,i+1, time_elapsed,avg_loss))
                accum_times.append(current_time-start_time)
                FILE.write("%.3f\n" % avg_loss)
                FILE.flush()
                running_loss = 0.0
    print("\nFinished Training\n")
    self.save_model(net)
    plt.figure(figsize=(10,5))
    plt.title("Training Loss vs. Iterations")
    plt.plot(training_loss_tally)
    plt.xlabel("iterations")
    plt.ylabel("training loss")
```

```
    plt.legend()
    plt.savefig("training_loss.png")
    #plt.show()


def run_code_for_testing_with_TEXTnetOrder2(self, net,
↪   hidden_size):
    net.load_state_dict(torch.load(self.dl_studio.path_saved_
    ↪   _model))
    classification_accuracy = 0.0
    negative_total = 0
    positive_total = 0
    confusion_matrix = torch.zeros(2,2)
    net.eval()
    with torch.no_grad():
        for i, data in enumerate(self.test_dataloader):
            cell_prev = net.initialize_cell(1)
            cell_prev_2_prev = net.initialize_cell(1)
            review_tensor,category,sentiment =
            ↪   data['review'], data['category'],
            ↪   data['sentiment']
            input = torch.zeros(1,review_tensor.shape[2])
            hidden = torch.zeros(1, hidden_size)
            for k in range(review_tensor.shape[1]):
                input[0,:] = review_tensor[0,k]
                output, hidden, cell = net(input, hidden,
                ↪   cell_prev_2_prev)
                if k == 0:
                    cell_prev = cell
                else:
                    cell_prev_2_prev = cell_prev
                    cell_prev = cell
            predicted_idx = torch.argmax(output).item()
            gt_idx = torch.argmax(sentiment).item()
            if i % 100 == 99:
                print("   [i=%4d]    predicted_label=%d
                ↪    gt_label=%d" % (i+1,
                ↪   predicted_idx,gt_idx))
            if predicted_idx == gt_idx:
                classification_accuracy += 1
            if gt_idx == 0:
                negative_total += 1
            elif gt_idx == 1:
                positive_total += 1
            confusion_matrix[gt_idx,predicted_idx] += 1
```

```
        print("\nOverall classification accuracy: %0.2f%%" %
        ↪  (float(classification_accuracy) * 100 /float(i)))
        out_percent = np.zeros((2,2), dtype='float')
        out_percent[0,0] = "%.3f" % (100 * confusion_matrix[0,0]
        ↪  / float(negative_total))
        out_percent[0,1] = "%.3f" % (100 * confusion_matrix[0,1]
        ↪  / float(negative_total))
        out_percent[1,0] = "%.3f" % (100 * confusion_matrix[1,0]
        ↪  / float(positive_total))
        out_percent[1,1] = "%.3f" % (100 * confusion_matrix[1,1]
        ↪  / float(positive_total))
        print("\n\nNumber of positive reviews tested: %d" %
        ↪  positive_total)
        print("\n\nNumber of negative reviews tested: %d" %
        ↪  negative_total)
        print("\n\nDisplaying the confusion matrix:\n")
        out_str = "                          "
        out_str +=  "%18s    %18s" % ('predicted negative',
        ↪  'predicted positive')
        print(out_str + "\n")
        for i,label in enumerate(['true negative', 'true
        ↪  positive']):
            out_str = "%12s:  " % label
            for j in range(2):
                out_str +=  "%18s" % out_percent[i,j]
            print(out_str)


class MyTEXTnetOrder2(nn.Module):

    # modified from corresponding code in DLStudio by Prof.Kak

    def __init__(self, input_size, hidden_size, output_size):
        super(MyTEXTnetOrder2, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.combined_to_hidden = nn.Linear(input_size +
        ↪  2*hidden_size, 2048)
        self.combined_to_hidden2 = nn.Linear(2048, hidden_size)
        self.combined_to_middle = nn.Linear(input_size +
        ↪  2*hidden_size,2048)
        self.middle_to_out1 = nn.Linear(2048,512)
        self.middle_to_out2 = nn.Linear(512,200)
        self.middle_to_out3 = nn.Linear(200, output_size)
```

```
        self.logsoftmax = nn.LogSoftmax(dim=1)
        self.dropout = nn.Dropout(p=0.2)
        # for the cell
        self.linear_for_cell = nn.Linear(hidden_size,
        ↪  hidden_size)
        self.relu = nn.PReLU()
        #self.embedding=nn.Embedding(vocab_size, 400)


        self.net1 = nn.Linear(2048,1024)
        self.net2 = nn.Linear(1024,2048)
    def forward(self, input, hidden, cell):

        out = input.clone()

        #h2c = torch.cat((hidden,hidden),1)
        h2c = self.relu(self.net1(hidden))
        h2c = self.net2(h2c)

        cell = h2c * cell

        combined = torch.cat((out, hidden, cell), 1) #
        ↪  input(B,1) # hidden(B,1024)


        hidden = self.relu(self.combined_to_hidden(combined))
        hidden = self.combined_to_hidden2(hidden)
        hidden = torch.tanh(hidden)


        out = self.combined_to_middle(combined)
        out = torch.nn.functional.relu(out)
        out = self.dropout(out)
        out = self.middle_to_out1(out)
        out = torch.nn.functional.relu(out)
        out = self.dropout(out)
        out = self.middle_to_out2(out)
        out = torch.nn.functional.relu(out)
        out = self.dropout(out)
        out = self.middle_to_out3(out)
        #out = self.logsoftmax(out)


        hidden_clone = hidden.clone()
#               cell =
↪  torch.tanh(self.linear_for_cell(hidden_clone))
```

```
        cell = torch.sigmoid(self.linear_for_cell(hidden_clone))
        #cell = self.relu(self.linear_for_cell(hidden_clone))

        return out,hidden,cell

    def initialize_cell(self, batch_size):
        weight = next(self.linear_for_cell.parameters()).data
        cell = weight.new(batch_size, self.hidden_size).zero_()
        return cell
```

```
dls = DLStudio(   dataroot
↪  ="./data/TextDatasets/sentiment_dataset/",
                  path_saved_model = "./saved_model",
                  momentum = 0.9,
                  learning_rate =  1e-5,
                  epochs = 5,
                  batch_size = 1,
                  classes = ('negative','positive'),
                  use_gpu = True,
              )

text_cl = MyTextClassification( dl_studio = dls )

dataserver_train = MySentimentAnalysisDataset(
                                train_or_test = 'train',
                                dl_studio = dls,
                               dataset_file = "sentiment_datase
                                ↪  t_train_40.tar.gz",
#                               dataset_file =
↪  "sentiment_dataset_train_200.tar.gz",
                  )
dataserver_test = MySentimentAnalysisDataset(
                                train_or_test = 'test',
                                dl_studio = dls,
                                dataset_file = "sentiment_datas
                                 ↪  et_test_40.tar.gz",
#                               dataset_file =
↪  "sentiment_dataset_test_200.tar.gz",
                  )
text_cl.dataserver_train = dataserver_train
text_cl.dataserver_test = dataserver_test
```

```
text_cl.load_SentimentAnalysisDataset(dataserver_train,
↪  dataserver_test)

vocab_size = dataserver_train.get_vocab_size()

model = MyTEXTnetOrder2(1, 2048, output_size=2)

number_of_learnable_params = sum(p.numel() for p in
↪  model.parameters() if p.requires_grad)
num_layers = len(list(model.parameters()))

print("\n\nThe number of layers in the model: %d" % num_layers)
print("\nThe number of learnable parameters in the model: %d" %
↪  number_of_learnable_params)

text_cl.run_code_for_training_with_TEXTnetOrder2(model,
↪  hidden_size=2048)
text_cl.run_code_for_testing_with_TEXTnetOrder2(model,
↪  hidden_size=2048)




# # use for baseline

# dls = DLStudio(  dataroot
↪  ="./data/TextDatasets/sentiment_dataset/",
#                   path_saved_model = "./saved_model",
#                   momentum = 0.9,
#                   learning_rate =  1e-5,
#                   epochs = 5 ,
#                   batch_size = 1,
#                   classes = ('negative','positive'),
#                   use_gpu = True,
#               )

# text_cl = MyTextClassification( dl_studio = dls )

# dataserver_train = MySentimentAnalysisDataset(
#                                   train_or_test = 'train',
#                                   dl_studio = dls,
#                                   dataset_file =
↪  "sentiment_dataset_train_40.tar.gz",
# #                                 dataset_file =
↪  "sentiment_dataset_train_200.tar.gz",
```

```
#                   )
# dataserver_test = MySentimentAnalysisDataset(
#                                     train_or_test = 'test',
#                                     dl_studio = dls,
#                                     dataset_file =
↪  "sentiment_dataset_test_40.tar.gz",
# #                                   dataset_file =
↪  "sentiment_dataset_train_200.tar.gz",
#                   )
# text_cl.dataserver_train = dataserver_train
# text_cl.dataserver_test = dataserver_test

# text_cl.load_SentimentAnalysisDataset(dataserver_train,
↪  dataserver_test)

# vocab_size = dataserver_train.get_vocab_size()

# model = text_cl.TEXTnetOrder2(1, hidden_size=512,
↪  output_size=2)

# number_of_learnable_params = sum(p.numel() for p in
↪  model.parameters() if p.requires_grad)
# num_layers = len(list(model.parameters()))

# print("\n\nThe number of layers in the model: %d" % num_layers)
# print("\nThe number of learnable parameters in the model: %d"
↪  % number_of_learnable_params)

# text_cl.run_code_for_training_with_TEXTnetOrder2(model,
↪  hidden_size=512)

# text_cl.run_code_for_testing_with_TEXTnetOrder2(model,
↪  hidden_size=512)
```

**hw07_task3.py**

```
# ECE 695DL 2021SPIRNG
# Homework7
# Zeyu Zhou
```

```
import random
import numpy
import torch
import torch.optim as optim
import torch.nn as nn
import copy
import math
from time import time
import gzip

import matplotlib.pyplot as plt
import os, sys

# I did not install DLStudio. I directly import from local
↪  python files. To run it on your device, comment out or
↪  modify this line.
sys.path.append('F:\ECE 695 Deep
↪  Learning\Homework\HW7\DLStudio-2.0.8\DLStudio')

seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)


##  watch -d -n 0.5 nvidia-smi

from DLStudio import *


class MySentimentAnalysisDataset(DLStudio.TextClassification.Sen
↪  timentAnalysisDataset):

    # modified from corresponding code in DLStudio by Prof.Kak

    def __init__(self, train_or_test, dl_studio, dataset_file):
        super(DLStudio.TextClassification.SentimentAnalysisDatas
        ↪  et,
        ↪  self).__init__()
        self.train_or_test = train_or_test
        root_dir = dl_studio.dataroot
```

```
f = gzip.open(root_dir + dataset_file, 'rb')
dataset = f.read()

# initialize max length
self.max_length = 0

if train_or_test == 'train':
    if sys.version_info[0] == 3:
        self.positive_reviews_train,
        ↪  self.negative_reviews_train, self.vocab =
        ↪  pickle.loads(dataset, encoding='latin1')
    else:
        self.positive_reviews_train,
        ↪  self.negative_reviews_train, self.vocab =
        ↪  pickle.loads(dataset)

    self.categories =
    ↪  sorted(list(self.positive_reviews_train.keys()))
    self.category_sizes_train_pos = {category :
    ↪  len(self.positive_reviews_train[category]) for
    ↪  category in self.categories}
    self.category_sizes_train_neg = {category :
    ↪  len(self.negative_reviews_train[category]) for
    ↪  category in self.categories}


    ##########################################################
    ↪  ##################################
    ################################## encode into
    ↪  integers ################################
    ##########################################################
    ↪  ##################################
    self.word2id = dict()
    i = 1
    for word in self.vocab:
        #self.word2id[word] = i/len(self.vocab)
        self.word2id[word] = i
        i += 1
    self.id2word = {i: word for word, i in
    ↪  self.word2id.items()}


    self.indexed_dataset_train = []
    for category in self.positive_reviews_train:
        for review in
        ↪  self.positive_reviews_train[category]:
```

```
            self.indexed_dataset_train.append([review,
            ↪  category, 1])
            if len(review) > self.max_length:
                self.max_length = len(review)
    for category in self.negative_reviews_train:
        for review in
        ↪  self.negative_reviews_train[category]:
            self.indexed_dataset_train.append([review,
            ↪  category, 0])
            if len(review) > self.max_length:
                self.max_length = len(review)
    random.shuffle(self.indexed_dataset_train)
elif train_or_test == 'test':
    if sys.version_info[0] == 3:
        self.positive_reviews_test,
        ↪  self.negative_reviews_test, self.vocab =
        ↪  pickle.loads(dataset, encoding='latin1')
    else:
        self.positive_reviews_test,
        ↪  self.negative_reviews_test, self.vocab =
        ↪  pickle.loads(dataset)
    self.vocab = sorted(self.vocab)
    self.categories =
    ↪  sorted(list(self.positive_reviews_test.keys()))
    self.category_sizes_test_pos = {category :
    ↪  len(self.positive_reviews_test[category]) for
    ↪  category in self.categories}
    self.category_sizes_test_neg = {category :
    ↪  len(self.negative_reviews_test[category]) for
    ↪  category in self.categories}

    ##################################################
    ↪  ################################
    ################################ encode into
    ↪  integers ##############################
    ##################################################
    ↪  ################################
    self.word2id = dict()
    i = 1
    for word in self.vocab:
        self.word2id[word] = i
        i += 1
    self.id2word = {i: word for word, i in
    ↪  self.word2id.items()}

    self.indexed_dataset_test = []
```

```
            for category in self.positive_reviews_test:
                for review in
                ↪  self.positive_reviews_test[category]:
                    self.indexed_dataset_test.append([review,
                    ↪  category, 1])
                    if len(review) > self.max_length:
                        self.max_length = len(review)
            for category in self.negative_reviews_test:
                for review in
                ↪  self.negative_reviews_test[category]:
                    self.indexed_dataset_test.append([review,
                    ↪  category, 0])
                    if len(review) > self.max_length:
                        self.max_length = len(review)
            random.shuffle(self.indexed_dataset_test)


    def review_to_integer_tensor(self, review):
        review_tensor = torch.zeros(self.max_length, 1)
        for i,word in enumerate(review):
            review_tensor[i,:] = self.word2id[word]
        return review_tensor

    def __getitem__(self, idx):
        sample = self.indexed_dataset_train[idx] if
        ↪  self.train_or_test == 'train' else
        ↪  self.indexed_dataset_test[idx]
        review = sample[0]
        review_category = sample[1]
        review_sentiment = sample[2]
        review_sentiment =
        ↪  self.sentiment_to_tensor(review_sentiment)
        review_tensor = self.review_to_integer_tensor(review)
        category_index = self.categories.index(review_category)
        sample = {'review'       : review_tensor,
                  'category'     : category_index, # should be
                  ↪  converted to tensor, but not yet used
                  'sentiment'    : review_sentiment }
        return sample



class MyTextClassification(DLStudio.TextClassification):

    # modified from corresponding code in DLStudio by Prof.Kak
```

```
def load_SentimentAnalysisDataset(self, dataserver_train,
↪  dataserver_test ):
    self.train_dataloader =
    ↪  torch.utils.data.DataLoader(dataserver_train,
                batch_size=self.dl_studio.batch_size,shuffle
                ↪  =True,
                ↪  num_workers=0)
    self.test_dataloader =
    ↪  torch.utils.data.DataLoader(dataserver_test,
                       batch_size=1,shuffle=False,
                       ↪  num_workers=0)

def run_code_for_training_with_TEXTnetOrder2(self, net,
↪  hidden_size):
    filename_for_out = "performance_numbers_" +
    ↪  str(self.dl_studio.epochs) + ".txt"
    FILE = open(filename_for_out, 'w')
    net = copy.deepcopy(net)
    net = net.to(self.dl_studio.device)
    ## Note that the TEXTnet and TEXTnetOrder2 both produce
    ↪  LogSoftmax output:
    #criterion = nn.NLLLoss()
    criterion = nn.MSELoss()
    accum_times = []
    optimizer = optim.SGD(net.parameters(),
                    lr=self.dl_studio.learning_rate,
                    ↪  momentum=self.dl_studio.momentum)
    start_time = time.perf_counter()
    training_loss_tally = []
    net.train()
    for epoch in range(self.dl_studio.epochs):
        print("")
        running_loss = 0.0
        for i, data in enumerate(self.train_dataloader):
            batch_size = data['review'].shape[0]
            cell_prev = net.initialize_cell(batch_size).to(s
            ↪  elf.dl_studio.device)
            cell_prev_2_prev = net.initialize_cell(batch_siz
            ↪  e).to(self.dl_studio.device)
            hidden = torch.zeros(batch_size, hidden_size)
            hidden = hidden.to(self.dl_studio.device)
            review_tensor,category,sentiment =
            ↪  data['review'], data['category'],
            ↪  data['sentiment']
```

```
            review_tensor =
            ↪  review_tensor.to(self.dl_studio.device)
            sentiment = sentiment.to(self.dl_studio.device)
            optimizer.zero_grad()
            input = torch.zeros(batch_size,review_tensor.sha⌋
            ↪  pe[2])
            input = input.to(self.dl_studio.device)
            for k in range(review_tensor.shape[1]):
                input = review_tensor[:,k]
                output, hidden, cell = net(input, hidden,
                ↪  cell_prev_2_prev)
                if k == 0:
                    cell_prev = cell
                else:
                    cell_prev_2_prev = cell_prev
                    cell_prev = cell
            #loss = criterion(output,
            ↪  torch.argmax(sentiment,1))
            loss =
            ↪  criterion(output.float(),sentiment.float())
            running_loss += loss.item()
            loss.backward()
            optimizer.step()
            #if i % 50 == 49:
            if i % 25 == 24:
                #avg_loss = running_loss / float(50)
                avg_loss = running_loss / float(25)
                training_loss_tally.append(avg_loss)
                current_time = time.perf_counter()
                time_elapsed = current_time-start_time
                print("[epoch:%d  iter:%4d  elapsed_time:
                ↪  %4d secs]     loss: %.5f" %
                ↪  (epoch+1,i+1, time_elapsed,avg_loss))
                accum_times.append(current_time-start_time)
                FILE.write("%.3f\n" % avg_loss)
                FILE.flush()
                running_loss = 0.0
    print("\nFinished Training\n")
    self.save_model(net)
    plt.figure(figsize=(10,5))
    plt.title("Training Loss vs. Iterations")
    plt.plot(training_loss_tally)
    plt.xlabel("iterations")
    plt.ylabel("training loss")
    plt.legend()
    plt.savefig("training_loss.png")
```

```
    #plt.show()

def run_code_for_testing_with_TEXTnetOrder2(self, net,
↪  hidden_size):
    net.load_state_dict(torch.load(self.dl_studio.path_saved_
    ↪  _model))
    classification_accuracy = 0.0
    negative_total = 0
    positive_total = 0
    confusion_matrix = torch.zeros(2,2)
    net.eval()
    with torch.no_grad():
        for i, data in enumerate(self.test_dataloader):
            batch_size = data['review'].shape[0]
            cell_prev = net.initialize_cell(batch_size)
            cell_prev_2_prev =
            ↪  net.initialize_cell(batch_size)
            review_tensor,category,sentiment =
            ↪  data['review'], data['category'],
            ↪  data['sentiment']
            input = torch.zeros(batch_size,review_tensor.sha_
            ↪  pe[2])
            hidden = torch.zeros(batch_size, hidden_size)
            for k in range(review_tensor.shape[1]):
                input = review_tensor[:,k]
                output, hidden, cell = net(input, hidden,
                ↪  cell_prev_2_prev)
                if k == 0:
                    cell_prev = cell
                else:
                    cell_prev_2_prev = cell_prev
                    cell_prev = cell
            predicted_idx = torch.argmax(output).item()
            gt_idx = torch.argmax(sentiment).item()
            if i % 100 == 99:
                print("   [i=%4d]    predicted_label=%d
                ↪   gt_label=%d" % (i+1,
                ↪  predicted_idx,gt_idx))
            if predicted_idx == gt_idx:
                classification_accuracy += 1
            if gt_idx == 0:
                negative_total += 1
            elif gt_idx == 1:
                positive_total += 1
            confusion_matrix[gt_idx,predicted_idx] += 1
```

```
        print("\nOverall classification accuracy: %0.2f%%" %
        ↪  (float(classification_accuracy) * 100 /float(i)))
        out_percent = np.zeros((2,2), dtype='float')
        out_percent[0,0] = "%.3f" % (100 * confusion_matrix[0,0]
        ↪  / float(negative_total))
        out_percent[0,1] = "%.3f" % (100 * confusion_matrix[0,1]
        ↪  / float(negative_total))
        out_percent[1,0] = "%.3f" % (100 * confusion_matrix[1,0]
        ↪  / float(positive_total))
        out_percent[1,1] = "%.3f" % (100 * confusion_matrix[1,1]
        ↪  / float(positive_total))
        print("\n\nNumber of positive reviews tested: %d" %
        ↪  positive_total)
        print("\n\nNumber of negative reviews tested: %d" %
        ↪  negative_total)
        print("\n\nDisplaying the confusion matrix:\n")
        out_str = "                          "
        out_str +=  "%18s    %18s" % ('predicted negative',
        ↪  'predicted positive')
        print(out_str + "\n")
        for i,label in enumerate(['true negative', 'true
        ↪  positive']):
            out_str = "%12s:  " % label
            for j in range(2):
                out_str +=  "%18s" % out_percent[i,j]
            print(out_str)



class MyTEXTnetOrder2(nn.Module):

    # modified from corresponding code in DLStudio by Prof.Kak

    def __init__(self, input_size, hidden_size, output_size):
        super(MyTEXTnetOrder2, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.combined_to_hidden = nn.Linear(input_size +
        ↪  2*hidden_size, 2048)
        self.combined_to_hidden2 = nn.Linear(2048, hidden_size)
        self.combined_to_middle = nn.Linear(input_size +
        ↪  2*hidden_size,2048)
        self.middle_to_out1 = nn.Linear(2048,512)
        self.middle_to_out2 = nn.Linear(512,200)
```

```
        self.middle_to_out3 = nn.Linear(200, output_size)
        self.logsoftmax = nn.LogSoftmax(dim=1)
        self.dropout = nn.Dropout(p=0.2)
        # for the cell
        self.linear_for_cell = nn.Linear(hidden_size,
        ↪  hidden_size)
        self.relu = nn.PReLU()
        #self.embedding=nn.Embedding(vocab_size, 400)


        self.net1 = nn.Linear(2048,1024)
        self.net2 = nn.Linear(1024,2048)
    def forward(self, input, hidden, cell):

        out = input.clone()

        #h2c = torch.cat((hidden,hidden),1)
        h2c = self.relu(self.net1(hidden))
        h2c = self.net2(h2c)

        cell = h2c * cell

        combined = torch.cat((out, hidden, cell), 1) #
        ↪  input(B,1) # hidden(B,1024)


        hidden = self.relu(self.combined_to_hidden(combined))
        hidden = self.combined_to_hidden2(hidden)
        hidden = torch.tanh(hidden)


        out = self.combined_to_middle(combined)
        out = torch.nn.functional.relu(out)
        out = self.dropout(out)
        out = self.middle_to_out1(out)
        out = torch.nn.functional.relu(out)
        out = self.dropout(out)
        out = self.middle_to_out2(out)
        out = torch.nn.functional.relu(out)
        out = self.dropout(out)
        out = self.middle_to_out3(out)
        #out = self.logsoftmax(out)


        hidden_clone = hidden.clone()
```

```
#                cell =
↪  torch.tanh(self.linear_for_cell(hidden_clone))
        cell = torch.sigmoid(self.linear_for_cell(hidden_clone))
        #cell = self.relu(self.linear_for_cell(hidden_clone))

        return out,hidden,cell

    def initialize_cell(self, batch_size):
        weight = next(self.linear_for_cell.parameters()).data
        cell = weight.new(batch_size, self.hidden_size).zero_()
        return cell
```

```
dls = DLStudio(   dataroot
↪  ="./data/TextDatasets/sentiment_dataset/",
                  path_saved_model = "./saved_model",
                  momentum = 0.9,
                  learning_rate =  1e-5,
                  epochs = 5,
                  batch_size = 20,
                  classes = ('negative','positive'),
                  use_gpu = True,
              )
```

```
text_cl = MyTextClassification( dl_studio = dls )

dataserver_train = MySentimentAnalysisDataset(
                                  train_or_test = 'train',
                                  dl_studio = dls,
                                 dataset_file = "sentiment_datase⌋
                                  ↪  t_train_40.tar.gz",
#                                 dataset_file =
↪  "sentiment_dataset_train_200.tar.gz",
                  )
dataserver_test = MySentimentAnalysisDataset(
                                  train_or_test = 'test',
                                  dl_studio = dls,
                                  dataset_file = "sentiment_datas⌋
                                   ↪  et_test_40.tar.gz",
```

```
#                                 dataset_file =
↪  "sentiment_dataset_test_200.tar.gz",
                  )
text_cl.dataserver_train = dataserver_train
text_cl.dataserver_test = dataserver_test

text_cl.load_SentimentAnalysisDataset(dataserver_train,
↪  dataserver_test)

vocab_size = dataserver_train.get_vocab_size()

model = MyTEXTnetOrder2(1, 2048, output_size=2)

number_of_learnable_params = sum(p.numel() for p in
↪  model.parameters() if p.requires_grad)
num_layers = len(list(model.parameters()))

print("\n\nThe number of layers in the model: %d" % num_layers)
print("\nThe number of learnable parameters in the model: %d" %
↪  number_of_learnable_params)

text_cl.run_code_for_training_with_TEXTnetOrder2(model,
↪  hidden_size=2048)
text_cl.run_code_for_testing_with_TEXTnetOrder2(model,
↪  hidden_size=2048)
```

# ECE 695DL Homework 7 Report

2021 SPRING

Zeyu Zhou
zhou1059@purdue.edu

**Note:** I run the code on Windows. To make it run on Windows, I have to make some necessary modification such as "/..." to "./..." and `num_workers` to 0. I don't test on Mac OS or Linux.
For all three tasks, I mainly modify some codes from DLStudio. I would point out which part I modify.

## 1 Task 0

I first start with a baseline. To be more specifc, I run `text_classification_with_TEXTnetOrder2.py` in DLStudio. Note that the baseline does not mean that I compare the performance of task0 with other three tasks. It's more like task0 is a starting point. I compare task1 with task0. Then I compare task2 with task1. Finally I compare task3 with task2.
I first run one epoch. See Figure 1 for train loss curve and some results.
Then I run five epochs. See Figure 2. We can observe that it does not actually converge. Though the overall accuracy is slightly higher, we can observe from the confusion matrix it actually predicts most reviews as positive.

## 2 Task 1

Main new modification: `MySentimentAnalysisDataset`

I use integer encoding here. Besides, I notice that the loss would be very large at first in this case. Hence, I do a normalization. I divide all integers by the length of word list so that all values would be within 0 and 1.
See Figure 3. Though the performance does not become better, we can observe that the number of parameters is much smaller. And the time of training decreases a lot. Then we have more freedom to try more complex model(there are many other things we can do such as run for more epochs or use more training samples...) with limited resources.

## 3 Task 2

Main new modification: `MyTEXTnetOrder2` `run_code_for_training_with_TEXTnetOrder2`, `run_code_for_testing_with_TEXTnetOrder2`

I add gating action and modify the architecture. Please read corresponding code for details.
Before talking about the results, I would like to describe the difficulty I have met here. Before trying to work on modifying the network, I try to find a setup that at least baseline model works. I have tried to modify the following things:

1. Use dataset-200 to train.
2. Try other loss function such as MSE Loss.
3. Decrease step size to 1e-7 or 1e-8.

4. Test with Professor Kak's GRU code.
5. Use one-hot encoding.
6. Remove integer normalization.
7. Change ReLU to PReLU.

The issue is that in many cases the loss would just oscillate around a certain point. Running more epochs does not lead to any improvement in performance. What's more, the probability of predicting positive and negative is always quite similar no matter what ground truth is. After investigating this for a long time, I still don't quite see what the main problem is. I try to print out the accuracy and confusion matrix in training and I can tell that it is as bad as test performance. I think two most probable issue is that the model capacity is not enough and some training setup may not work well.
Anyway, finally I decide to use the following setup. (Task 3 will follow the setup here)

1. I begin to use Adam optimizer.
2. I keep learning rate as 1e-5.
3. I use MSE Loss.
4. I remove the normalization of integer encoding.
5. I run 5 epochs and use dataset-40.
6. I add `net.train()` and `net.eval()` to the code.
7. I try to add more learnable parameters in the model.

See Figure 4 for results with original model.
See Figure 5 for results with my model. We can observe that though training time is much longer, there is an obvious improvement in overall test accuracy and confusion matrix.

# 4 Task 3

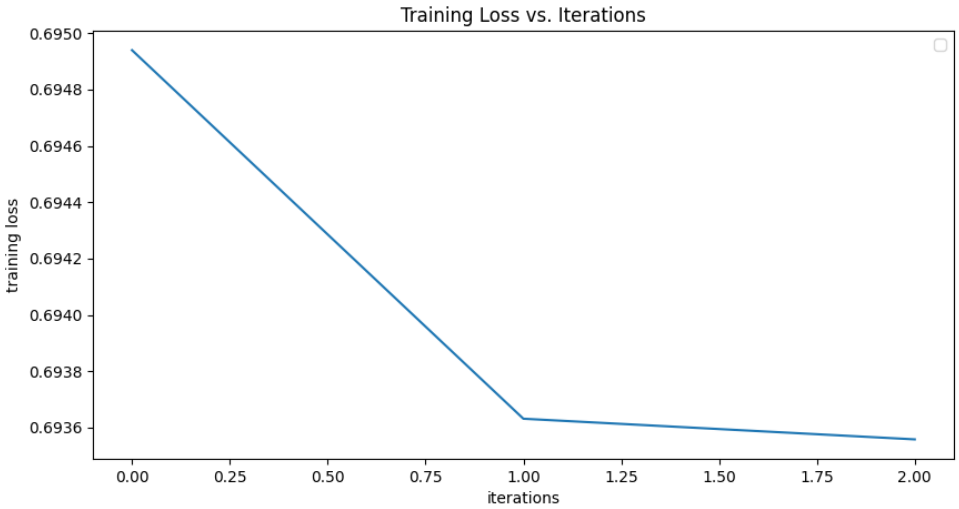
Main new modification: `MySentimentAnalysisDataset`, `load_SentimentAnalysisDataset`, `run_code_for_training_with_TEXTnetOrder2`, `run_code_for_testing_with_TEXTnetOrder2`

I use padding strategy here i.e. find the length of longest review and set it as the length of all reviews(fill with 0 if there are less words). A necessary modification is that I start assigning integers from 1 instead of 0 as task 1 to avoid the case where "0" represents a word.
Besides, I also modified training/testing codes in DLStudio because the original codes have some hard-coding which is not compatible with larger batch size($> 1$).
It is not fair to compare the performance directly with that in task 2 since use padding itself can make some difference to the performance. However, running with `batch_size = 1` using my model on my PC can take around 2000 seconds to run only 500 iterations. Besides, when running it on Colab, I do observe that the training actually becomes more unstable with `batch_size = 1`. It is as expected because it is still unclear how padding(not batching) will influence the training. See Figure 8 for some records.
I also try to use the original model with `batch_size = 1`. See Figure 9 for some records.

The issue might be fixed after adjusting some experiment setup such as loss function. However since the main goal of this task is to see how batching will affect the experiment, I did not do those tests.

See Figure 6 for results with `batch_size = 10` using my model. Apparently now the loss does not blow up which implies much more stable training. According to Prof. Bouman's notes, this is the benefit we should expect. Besides, the training is much faster.

See Figure 7 for results with `batch_size = 20` using my model. We can observe that larger batch size significantly decrease the training time. The train loss seems to be more smooth but it definitely needs more carefully designed experiments to give such a conclusion.

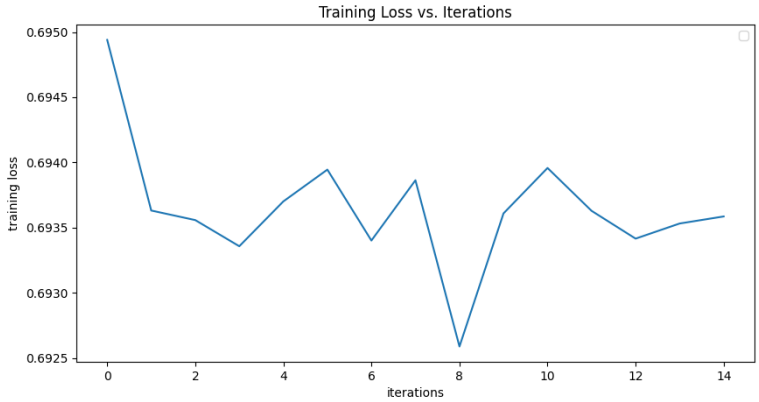



(a) Train Loss

```
The number of layers in the model: 8

The number of learnable parameters in the model: 11294770

The size of the vocabulary (which is also the size of the one-hot vecs for words): 17001



[epoch:1  iter: 500  elapsed_time:   57 secs]     loss: 0.69494
[epoch:1  iter:1000  elapsed_time:  107 secs]     loss: 0.69363
[epoch:1  iter:1500  elapsed_time:  159 secs]     loss: 0.69356

Finished Training

No handles with labels found to put in legend.
   [i= 100]    predicted_label=0       gt_label=0
   [i= 200]    predicted_label=1       gt_label=0
   [i= 300]    predicted_label=0       gt_label=1

Overall classification accuracy: 50.00%


Number of positive reviews tested: 200


Number of negative reviews tested: 195


Displaying the confusion matrix:

                     predicted negative    predicted positive

true negative:              69.744              30.256
true positive:                69.5                30.5
```

(b) result

Figure 1: Task0: 1 Epoch

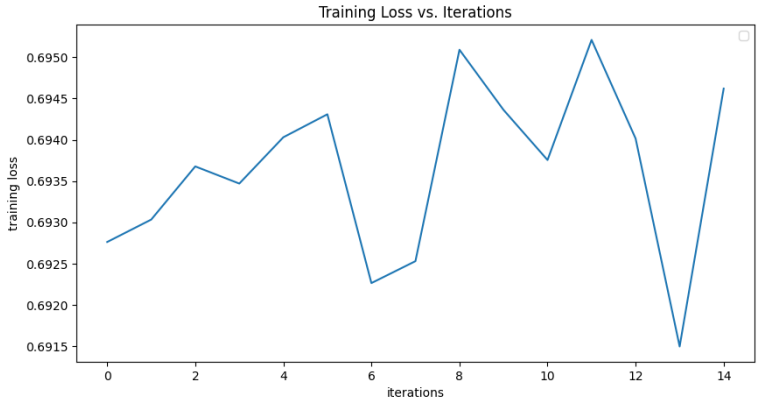



(a) Train Loss

```
The number of layers in the model: 8

The number of learnable parameters in the model: 11294770

The size of the vocabulary (which is also the size of the one-hot vecs for words): 17001



[epoch:1  iter: 500  elapsed_time:   57 secs]     loss: 0.69494
[epoch:1  iter:1000  elapsed_time:  107 secs]     loss: 0.69363
[epoch:1  iter:1500  elapsed_time:  158 secs]     loss: 0.69356

[epoch:2  iter: 500  elapsed_time:  220 secs]     loss: 0.69336
[epoch:2  iter:1000  elapsed_time:  272 secs]     loss: 0.69370
[epoch:2  iter:1500  elapsed_time:  324 secs]     loss: 0.69395

[epoch:3  iter: 500  elapsed_time:  382 secs]     loss: 0.69340
[epoch:3  iter:1000  elapsed_time:  437 secs]     loss: 0.69386
[epoch:3  iter:1500  elapsed_time:  494 secs]     loss: 0.69259

[epoch:4  iter: 500  elapsed_time:  552 secs]     loss: 0.69361
[epoch:4  iter:1000  elapsed_time:  606 secs]     loss: 0.69396
[epoch:4  iter:1500  elapsed_time:  661 secs]     loss: 0.69363

[epoch:5  iter: 500  elapsed_time:  724 secs]     loss: 0.69342
[epoch:5  iter:1000  elapsed_time:  777 secs]     loss: 0.69353
[epoch:5  iter:1500  elapsed_time:  829 secs]     loss: 0.69359

Finished Training

No handles with labels found to put in legend.
   [i= 100]    predicted_label=1       gt_label=0
   [i= 200]    predicted_label=1       gt_label=0
   [i= 300]    predicted_label=1       gt_label=1

Overall classification accuracy: 50.51%


Number of positive reviews tested: 200


Number of negative reviews tested: 195


Displaying the confusion matrix:

                    predicted negative    predicted positive

true negative:              0.513             99.487
true positive:                1.0               99.0
```

(b) result

Figure 2: Task0: 5 Epochs

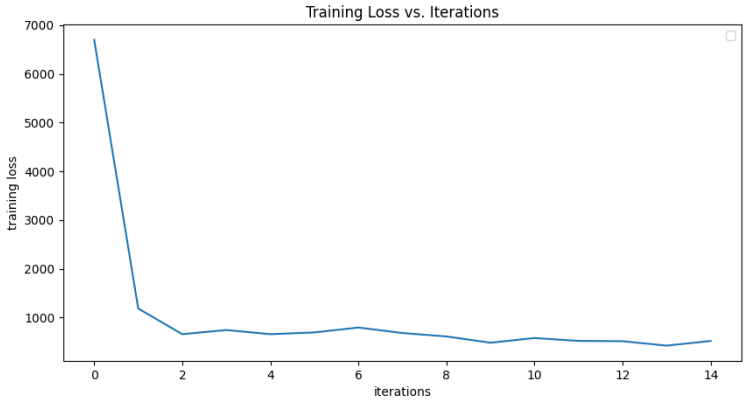



(a) Train Loss

```
The number of layers in the model: 8

The number of learnable parameters in the model: 890770

[epoch:1  iter: 500  elapsed_time:   27 secs]     loss: 0.69276
[epoch:1  iter:1000  elapsed_time:   52 secs]     loss: 0.69303
[epoch:1  iter:1500  elapsed_time:   78 secs]     loss: 0.69368

[epoch:2  iter: 500  elapsed_time:  107 secs]     loss: 0.69347
[epoch:2  iter:1000  elapsed_time:  133 secs]     loss: 0.69403
[epoch:2  iter:1500  elapsed_time:  160 secs]     loss: 0.69431

[epoch:3  iter: 500  elapsed_time:  190 secs]     loss: 0.69227
[epoch:3  iter:1000  elapsed_time:  215 secs]     loss: 0.69253
[epoch:3  iter:1500  elapsed_time:  242 secs]     loss: 0.69509

[epoch:4  iter: 500  elapsed_time:  274 secs]     loss: 0.69436
[epoch:4  iter:1000  elapsed_time:  300 secs]     loss: 0.69376
[epoch:4  iter:1500  elapsed_time:  325 secs]     loss: 0.69521

[epoch:5  iter: 500  elapsed_time:  354 secs]     loss: 0.69402
[epoch:5  iter:1000  elapsed_time:  381 secs]     loss: 0.69150
[epoch:5  iter:1500  elapsed_time:  407 secs]     loss: 0.69462

Finished Training

No handles with labels found to put in legend.
   [i= 100]    predicted_label=1       gt_label=0
   [i= 200]    predicted_label=1       gt_label=0
   [i= 300]    predicted_label=1       gt_label=1

Overall classification accuracy: 50.00%


Number of positive reviews tested: 200


Number of negative reviews tested: 195


Displaying the confusion matrix:

                    predicted negative    predicted positive

true negative:              20.0                  80.0
true positive:              21.0                  79.0
```

(b) result

Figure 3: Task1: 5 Epochs

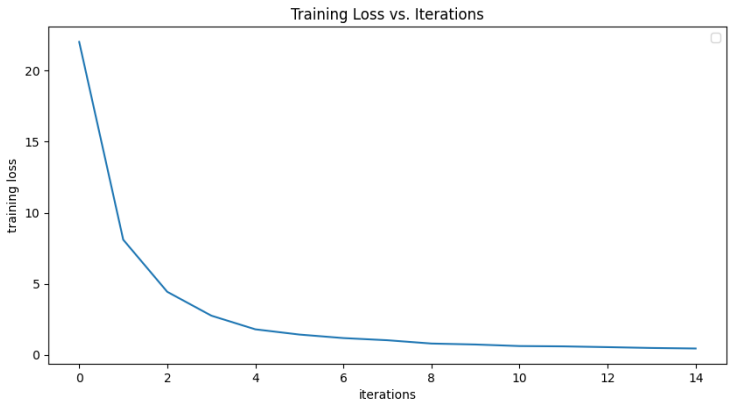



(a) Train Loss

```
The number of layers in the model: 8

The number of learnable parameters in the model: 890770

[epoch:1  iter: 500  elapsed_time:   28 secs]     loss: 6701.27168
[epoch:1  iter:1000  elapsed_time:   54 secs]     loss: 1186.86242
[epoch:1  iter:1500  elapsed_time:   81 secs]     loss: 659.97586

[epoch:2  iter: 500  elapsed_time:  111 secs]     loss: 745.00067
[epoch:2  iter:1000  elapsed_time:  137 secs]     loss: 659.89840
[epoch:2  iter:1500  elapsed_time:  165 secs]     loss: 695.69381

[epoch:3  iter: 500  elapsed_time:  196 secs]     loss: 796.73661
[epoch:3  iter:1000  elapsed_time:  221 secs]     loss: 683.45784
[epoch:3  iter:1500  elapsed_time:  249 secs]     loss: 613.18514

[epoch:4  iter: 500  elapsed_time:  281 secs]     loss: 484.58488
[epoch:4  iter:1000  elapsed_time:  308 secs]     loss: 580.34105
[epoch:4  iter:1500  elapsed_time:  333 secs]     loss: 522.13459

[epoch:5  iter: 500  elapsed_time:  363 secs]     loss: 515.41974
[epoch:5  iter:1000  elapsed_time:  390 secs]     loss: 424.89784
[epoch:5  iter:1500  elapsed_time:  418 secs]     loss: 520.98147

Finished Training

No handles with labels found to put in legend.
   [i= 100]    predicted_label=0       gt_label=0
   [i= 200]    predicted_label=1       gt_label=0
   [i= 300]    predicted_label=1       gt_label=1

Overall classification accuracy: 50.76%


Number of positive reviews tested: 200


Number of negative reviews tested: 195


Displaying the confusion matrix:

                    predicted negative    predicted positive

true negative:              34.359              65.641
true positive:                33.5                66.5
```

(b) result

Figure 4: Task2: Original Model

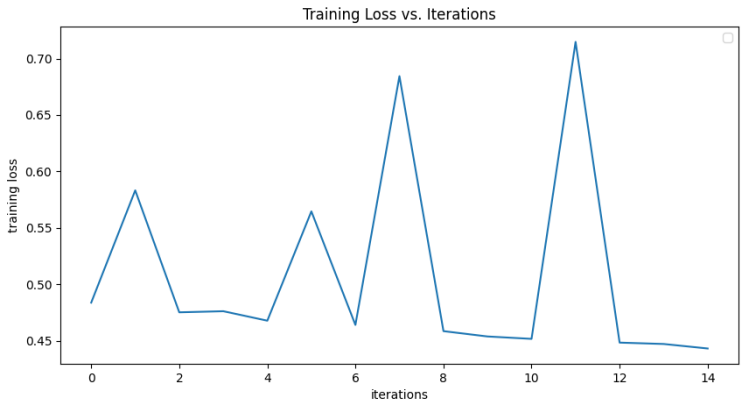



(a) Train Loss

```
The number of layers in the model: 19

The number of learnable parameters in the model: 30527579

[epoch:1  iter: 500  elapsed_time:  128 secs]     loss: 22.00668
[epoch:1  iter:1000  elapsed_time:  268 secs]     loss: 8.10679
[epoch:1  iter:1500  elapsed_time:  394 secs]     loss: 4.44377

[epoch:2  iter: 500  elapsed_time:  552 secs]     loss: 2.76999
[epoch:2  iter:1000  elapsed_time:  676 secs]     loss: 1.80825
[epoch:2  iter:1500  elapsed_time:  811 secs]     loss: 1.44088

[epoch:3  iter: 500  elapsed_time:  962 secs]     loss: 1.19315
[epoch:3  iter:1000  elapsed_time: 1099 secs]     loss: 1.04092
[epoch:3  iter:1500  elapsed_time: 1235 secs]     loss: 0.80912

[epoch:4  iter: 500  elapsed_time: 1420 secs]     loss: 0.74154
[epoch:4  iter:1000  elapsed_time: 1582 secs]     loss: 0.63322
[epoch:4  iter:1500  elapsed_time: 1749 secs]     loss: 0.61088

[epoch:5  iter: 500  elapsed_time: 1941 secs]     loss: 0.55725
[epoch:5  iter:1000  elapsed_time: 2097 secs]     loss: 0.49779
[epoch:5  iter:1500  elapsed_time: 2257 secs]     loss: 0.46146

Finished Training

No handles with labels found to put in legend.
   [i= 100]    predicted_label=0       gt_label=0
   [i= 200]    predicted_label=1       gt_label=0
   [i= 300]    predicted_label=1       gt_label=1

Overall classification accuracy: 53.81%


Number of positive reviews tested: 200


Number of negative reviews tested: 195


Displaying the confusion matrix:

                    predicted negative    predicted positive

true negative:              54.359              45.641
true positive:                47.0                53.0
```

(b) result

Figure 5: Task2: My model

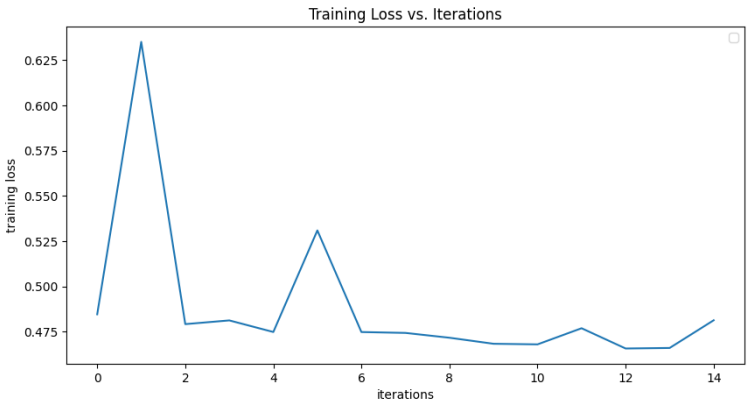



(a) Train Loss

```
The number of layers in the model: 19

The number of learnable parameters in the model: 30527579

[epoch:1  iter:  50  elapsed_time:  188 secs]     loss: 0.48366
[epoch:1  iter: 100  elapsed_time:  376 secs]     loss: 0.58320
[epoch:1  iter: 150  elapsed_time:  561 secs]     loss: 0.47510

[epoch:2  iter:  50  elapsed_time:  778 secs]     loss: 0.47608
[epoch:2  iter: 100  elapsed_time:  967 secs]     loss: 0.46769
[epoch:2  iter: 150  elapsed_time: 1153 secs]     loss: 0.56457

[epoch:3  iter:  50  elapsed_time: 1375 secs]     loss: 0.46393
[epoch:3  iter: 100  elapsed_time: 1564 secs]     loss: 0.68443
[epoch:3  iter: 150  elapsed_time: 1753 secs]     loss: 0.45842

[epoch:4  iter:  50  elapsed_time: 1971 secs]     loss: 0.45370
[epoch:4  iter: 100  elapsed_time: 2161 secs]     loss: 0.45157
[epoch:4  iter: 150  elapsed_time: 2351 secs]     loss: 0.71479

[epoch:5  iter:  50  elapsed_time: 2571 secs]     loss: 0.44826
[epoch:5  iter: 100  elapsed_time: 2760 secs]     loss: 0.44704
[epoch:5  iter: 150  elapsed_time: 2950 secs]     loss: 0.44304

Finished Training

No handles with labels found to put in legend.
   [i= 100]    predicted_label=0       gt_label=0
   [i= 200]    predicted_label=0       gt_label=0
   [i= 300]    predicted_label=0       gt_label=1

Overall classification accuracy: 49.75%


Number of positive reviews tested: 200


Number of negative reviews tested: 195


Displaying the confusion matrix:

                    predicted negative    predicted positive

true negative:              100.0                 0.0
true positive:               99.5                 0.5
```

(b) result

Figure 6: Task3: `batch_size=10`




(a) Train Loss

```
The number of layers in the model: 19

The number of learnable parameters in the model: 30527579

[epoch:1  iter:  25  elapsed_time:   96 secs]     loss: 0.48461
[epoch:1  iter:  50  elapsed_time:  194 secs]     loss: 0.63510
[epoch:1  iter:  75  elapsed_time:  291 secs]     loss: 0.47919

[epoch:2  iter:  25  elapsed_time:  403 secs]     loss: 0.48124
[epoch:2  iter:  50  elapsed_time:  500 secs]     loss: 0.47484
[epoch:2  iter:  75  elapsed_time:  597 secs]     loss: 0.53095

[epoch:3  iter:  25  elapsed_time:  708 secs]     loss: 0.47482
[epoch:3  iter:  50  elapsed_time:  804 secs]     loss: 0.47433
[epoch:3  iter:  75  elapsed_time:  900 secs]     loss: 0.47167

[epoch:4  iter:  25  elapsed_time: 1012 secs]     loss: 0.46834
[epoch:4  iter:  50  elapsed_time: 1107 secs]     loss: 0.46801
[epoch:4  iter:  75  elapsed_time: 1203 secs]     loss: 0.47692

[epoch:5  iter:  25  elapsed_time: 1315 secs]     loss: 0.46573
[epoch:5  iter:  50  elapsed_time: 1411 secs]     loss: 0.46602
[epoch:5  iter:  75  elapsed_time: 1506 secs]     loss: 0.48132

Finished Training

No handles with labels found to put in legend.
   [i= 100]    predicted_label=0       gt_label=0
   [i= 200]    predicted_label=0       gt_label=0
   [i= 300]    predicted_label=0       gt_label=1

Overall classification accuracy: 49.75%


Number of positive reviews tested: 200


Number of negative reviews tested: 195


Displaying the confusion matrix:

                    predicted negative    predicted positive

true negative:              100.0                 0.0
true positive:               99.5                 0.5
```

(b) result

Figure 7: Task3: `batch_size=20`

```
The number of layers in the model: 19

The number of learnable parameters in the model: 30527579

[epoch:1  iter: 500  elapsed_time: 1161 secs]    loss: 0.52233
[epoch:1  iter:1000  elapsed_time: 2322 secs]    loss: 0.48973
[epoch:1  iter:1500  elapsed_time: 3483 secs]    loss: 0.46626

[epoch:2  iter: 500  elapsed_time: 4828 secs]    loss: 0.43045
[epoch:2  iter:1000  elapsed_time: 5991 secs]    loss: 2.11701
[epoch:2  iter:1500  elapsed_time: 7154 secs]    loss: 0.45220

[epoch:3  iter: 500  elapsed_time: 8498 secs]    loss: 0.41553
[epoch:3  iter:1000  elapsed_time: 9654 secs]    loss: 0.39066
[epoch:3  iter:1500  elapsed_time: 10810 secs]    loss: nan

[epoch:4  iter: 500  elapsed_time: 12149 secs]    loss: nan
[epoch:4  iter:1000  elapsed_time: 13303 secs]    loss: nan
[epoch:4  iter:1500  elapsed_time: 14456 secs]    loss: nan

[epoch:5  iter: 500  elapsed_time: 15789 secs]    loss: nan
[epoch:5  iter:1000  elapsed_time: 16940 secs]    loss: nan
[epoch:5  iter:1500  elapsed_time: 18089 secs]    loss: nan
```

(a) Train Loss

Figure 8: Task3: Pre experiment on Colab using my model

```
[epoch:2  iter:  50  elapsed_time: 1238 secs]    loss: 1.67854
[epoch:2  iter: 100  elapsed_time: 1276 secs]    loss: 1.67530
[epoch:2  iter: 150  elapsed_time: 1314 secs]    loss: 1.67244
[epoch:2  iter: 200  elapsed_time: 1352 secs]    loss: 1.67220
[epoch:2  iter: 250  elapsed_time: 1390 secs]    loss: 1.67725
[epoch:2  iter: 300  elapsed_time: 1428 secs]    loss: 1.67893
[epoch:2  iter: 350  elapsed_time: 1465 secs]    loss: 1.67198
[epoch:2  iter: 400  elapsed_time: 1503 secs]    loss: 1.67187
[epoch:2  iter: 450  elapsed_time: 1541 secs]    loss: 162244732893.45728
[epoch:2  iter: 500  elapsed_time: 1578 secs]    loss: 372603.52789
[epoch:2  iter: 550  elapsed_time: 1616 secs]    loss: 472521.46563
[epoch:2  iter: 600  elapsed_time: 1654 secs]    loss: 463902.86125
[epoch:2  iter: 650  elapsed_time: 1692 secs]    loss: 454608.23250
[epoch:2  iter: 700  elapsed_time: 1730 secs]    loss: 445570.43688
[epoch:2  iter: 750  elapsed_time: 1767 secs]    loss: 436842.63062
[epoch:2  iter: 800  elapsed_time: 1805 secs]    loss: 428047.24500
[epoch:2  iter: 850  elapsed_time: 1843 secs]    loss: 419774.28063
[epoch:2  iter: 900  elapsed_time: 1881 secs]    loss: 411392.38250
[epoch:2  iter: 950  elapsed_time: 1919 secs]    loss: 403069.80625
[epoch:2  iter:1000  elapsed_time: 1957 secs]    loss: 395073.44188
[epoch:2  iter:1050  elapsed_time: 1995 secs]    loss: 387095.41063
[epoch:2  iter:1100  elapsed_time: 2034 secs]    loss: 379606.75000
[epoch:2  iter:1150  elapsed_time: 2072 secs]    loss: 372161.85375
```

(a) Train Loss

```
[epoch:3  iter:  50  elapsed_time: 2439 secs]    loss: 307193.33875
[epoch:3  iter: 100  elapsed_time: 2478 secs]    loss: 300975.47000
[epoch:3  iter: 150  elapsed_time: 2517 secs]    loss: 295004.40000
[epoch:3  iter: 200  elapsed_time: 2556 secs]    loss: 525091810411472.75000
[epoch:3  iter: 250  elapsed_time: 2595 secs]    loss: 991465717.76000
[epoch:3  iter: 300  elapsed_time: 2634 secs]    loss: 977150728.96000
[epoch:3  iter: 350  elapsed_time: 2673 secs]    loss: 957799655.68000
[epoch:3  iter: 400  elapsed_time: 2711 secs]    loss: 938792295.68000
[epoch:3  iter: 450  elapsed_time: 2750 secs]    loss: 920171303.68000
[epoch:3  iter: 500  elapsed_time: 2789 secs]    loss: 901922350.08000
[epoch:3  iter: 550  elapsed_time: 2827 secs]    loss: 884021926.40000
[epoch:3  iter: 600  elapsed_time: 2866 secs]    loss: 866480096.00000
[epoch:3  iter: 650  elapsed_time: 2904 secs]    loss: 849292684.80000
[epoch:3  iter: 700  elapsed_time: 2942 secs]    loss: 832441336.32000
[epoch:3  iter: 750  elapsed_time: 2980 secs]    loss: 815930063.36000
[epoch:3  iter: 800  elapsed_time: 3019 secs]    loss: 799740702.72000
[epoch:3  iter: 850  elapsed_time: 3057 secs]    loss: 783874117.12000
[epoch:3  iter: 900  elapsed_time: 3095 secs]    loss: 768322965.76000
[epoch:3  iter: 950  elapsed_time: 3134 secs]    loss: 753082073.60000
[epoch:3  iter:1000  elapsed_time: 3172 secs]    loss: 738145843.20000
[epoch:3  iter:1050  elapsed_time: 3211 secs]    loss: 723500435.20000
```

(b) Train Loss

Figure 9: Task3: Pre experiment on Colab using original model