BME 646/ ECE695DL: Homework 8

Varun Aggarwal

April 26, 2022

1 Introduction

This assignment was aimed at implementing Gated Recurrent Unit (GRU) network logic. In addition, training of two implementations of GRU (PyTorch and Dr. Kak's pmGRU ¡poor man's GRU¿)had to be demonstrated for sentiment analysis.

2 Methodology

This assignment was well defined. Majority of the training code came from DLStudio V2.2.2.My contribution was designing a network that works both with PyTorch's implementation of GRU and Dr. Kak's variation on GRU also called pmGRU.

List of tasks are as follows:

1. Training PyTorch GRU inspired model
2. Creating and training Dr. Kak's GRU inspired model
3. Comparing results of two implementations

Task 1

Implementing torch GRU had few challenges. Most of them were relating to word2vec download and PyTorch version. The underlying model was used from DLStudio library. A training file was used to initiate training of the model. After successful training, a testing script was used to evaluate model's performance. The default network parameters were insufficient to train the model. In fact, they lead to divergence. By increasing the learning rate to 1e-3 the problem was avoided. The result of training i.e. the loss curve for 3 epochs is given in Figure 1.

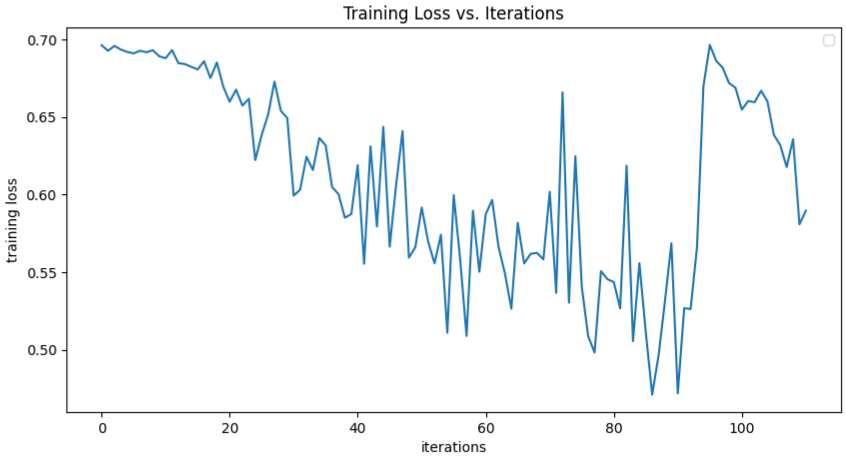

Figure 1: Loss curve for PyTorch GRU network

Task 2

Implementing Dr. Kak's pmGRU required few modifications to Task1 implementation. It includes, changing the hidden layer dimension, changing output_size, adjusting learning rate and modifying last layer to incorporate softmax function.Word2vec was used instead of fasttext. A training file was used to initiate training of the model. After successful training, a testing script was used to evaluate model's performance. The learning rate was set to 1e-3. The result of training i.e. the loss curve for 3 epochs is given in Figure 2.

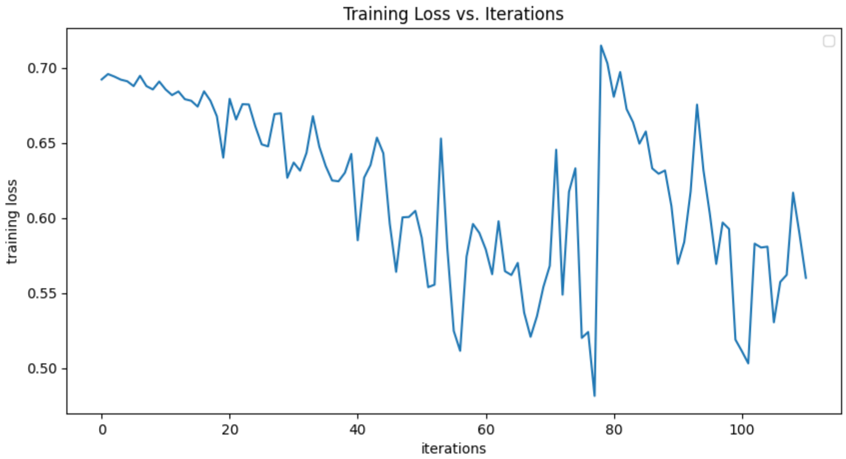

Figure 2: Loss curve for pmGRU network

Task 3

In the final task, trained models from Task 1 and 2 were evaluated on a testing dataset. The performance matrix of choice was confusion matrix. The results indicate that pmGRU lead to higher performance. Although, this conclusion is contingent on the fact that extensive hyper-parameter search was not performed. With a different set of hyper-parameters, the results could be different.

The confusion matrix for GRU implementation is given in Table 1 and for pmGRU implementation is given in Table 2.

Table 1: Confusion Matrix for GRU model

	Predicted Negative	**Predicted Positive**
True Negative	23.86 %	76.14 %
True Positive	3.47 %	96.53 %

Table 2: Confusion Matrix for pmGRU model

	Predicted Negative	**Predicted Positive**
True Negative	68.00 %	32.00 %
True Positive	15.20 %	84.80 %

3 Implementation

CODE-hw08_training.py

```
"""
Homework 8: Create GRU network
Author: Varun Aggarwal
Last Modified: 25 Apr 2022
Modifed from DLStudioV2.2.2
"""

import random
import numpy
import torch
import os, sys


"""
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)
"""


sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.2.2")
from DLStudio import *
from DataPrediction import *
import modelpmGRU as pmGRU

## USER DEFINED: parameters
type_GRU = "torch"
# type_GRU = "pmGRU"
dataroot = "/home/varun/work/courses/why2learn/hw/DLStudio-2.2.2/Examples/
   ↪ data/"
dataset_archive_train = "sentiment_dataset_train_200.tar.gz"
dataset_archive_test = "sentiment_dataset_test_200.tar.gz"
path_to_saved_embeddings = "/home/varun/work/courses/why2learn/hw/DLStudio
```

```
    ↪ -2.2.2/Examples/runs/"

if type_GRU == "torch":
    dls = DLStudio(
                  dataroot = dataroot,
                  path_saved_model = "/home/varun/work/courses/why2learn/hw/
                      ↪ hw8/runs/saved_model_GRU.pt",
                  momentum = 0.9,
                  learning_rate = 1e-3,
                  epochs = 3,
                  batch_size = 1,
                  classes = ('negative','positive'),
                  use_gpu = True,
              )
elif type_GRU=="pmGRU":
    dls = DLStudio(
                  dataroot = dataroot,
                  path_saved_model = "/home/varun/work/courses/why2learn/hw/
                      ↪ hw8/runs/saved_model_pmGRU.pt",
                  momentum = 0.9,
                  learning_rate = 1e-3,
                  epochs = 3,
                  batch_size = 1,
                  classes = ('negative','positive'),
                  use_gpu = True,
              )


text_cl = DLStudio.TextClassificationWithEmbeddings( dl_studio = dls )

dataserver_train = DLStudio.TextClassificationWithEmbeddings.
    ↪ SentimentAnalysisDataset(
                            train_or_test = 'train',
                            dl_studio = dls,
                            dataset_file = dataset_archive_train,
                            path_to_saved_embeddings =
                                ↪ path_to_saved_embeddings,
              )
dataserver_test = DLStudio.TextClassificationWithEmbeddings.
    ↪ SentimentAnalysisDataset(
                            train_or_test = 'test',
                            dl_studio = dls,
                             dataset_file = dataset_archive_test,
                             path_to_saved_embeddings =
                                ↪ path_to_saved_embeddings,
              )
text_cl.dataserver_train = dataserver_train
text_cl.dataserver_test = dataserver_test

text_cl.load_SentimentAnalysisDataset(dataserver_train, dataserver_test)

if type_GRU=="torch":
    model = text_cl.GRUnetWithEmbeddings(input_size=300, hidden_size=100,
       ↪ output_size=2, num_layers=2)
elif type_GRU=="pmGRU":
      model = pmGRU.custom_pmGRU(input_size=300, hidden_size=100,
         ↪ output_size=2, batch_size=dls.batch_size, num_layers=1)

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)

## TRAINING:
print("\nStarting␣training\n")
text_cl.run_code_for_training_for_text_classification_with_GRU_word2vec(
   ↪ model, display_train_loss=True)
```

CODE-hw08_testing.py

```
"""
Homework 8: Create GRU network
Author: Varun Aggarwal
Last Modified: 25 Apr 2022
Modifed from DLStudioV2.2.2
"""

import random
import numpy
import torch
import os, sys


"""
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)
"""


sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.2.2")
from DLStudio import *
from DataPrediction import *
import modelpmGRU as pmGRU

type_GRU = "torch"
# type_GRU = "pmGRU"


dataroot = "/home/varun/work/courses/why2learn/hw/DLStudio-2.2.2/Examples/
   ↪ data/"

dataset_archive_train = "sentiment_dataset_train_200.tar.gz"
dataset_archive_test = "sentiment_dataset_test_200.tar.gz"

path_to_saved_embeddings = "/home/varun/work/courses/why2learn/hw/DLStudio
   ↪ -2.2.2/Examples/runs/"

if type_GRU == "torch":
    dls = DLStudio(
                    dataroot = dataroot,
                    path_saved_model = "/home/varun/work/courses/why2learn/hw/
                       ↪ hw8/runs/saved_model_GRU.pt",
                    momentum = 0.9,
                    learning_rate = 1e-3,
                    epochs = 3,
                    batch_size = 1,
                    classes = ('negative','positive'),
                    use_gpu = True,
                )
elif type_GRU=="pmGRU":
    dls = DLStudio(
                    dataroot = dataroot,
                    path_saved_model = "/home/varun/work/courses/why2learn/hw/
                       ↪ hw8/runs/saved_model_pmGRU.pt",
                    momentum = 0.9,
                    learning_rate = 1e-3,
                    epochs = 3,
                    batch_size = 1,
                    classes = ('negative','positive'),
                    use_gpu = True,
                )


text_cl = DLStudio.TextClassificationWithEmbeddings( dl_studio = dls )

dataserver_train = DLStudio.TextClassificationWithEmbeddings.
   ↪ SentimentAnalysisDataset(
                             train_or_test = 'train',
                             dl_studio = dls,
                             dataset_file = dataset_archive_train,
                             path_to_saved_embeddings =
                                ↪ path_to_saved_embeddings,
                )
dataserver_test = DLStudio.TextClassificationWithEmbeddings.
   ↪ SentimentAnalysisDataset(
                             train_or_test = 'test',
                             dl_studio = dls,
                             dataset_file = dataset_archive_test,
                             path_to_saved_embeddings =
                                ↪ path_to_saved_embeddings,
                )
text_cl.dataserver_train = dataserver_train
text_cl.dataserver_test = dataserver_test

text_cl.load_SentimentAnalysisDataset(dataserver_train, dataserver_test)

if type_GRU=="torch":
    model = text_cl.GRUnetWithEmbeddings(input_size=300, hidden_size=100,
       ↪ output_size=2, num_layers=2)
elif type_GRU=="pmGRU":
    model = pmGRU.custom_pmGRU(input_size=300, hidden_size=100, output_size
       ↪ =2, batch_size=dls.batch_size, num_layers=1)


## TESTING:
text_cl.run_code_for_testing_text_classification_with_GRU_word2vec(model)
```

CODE-model-pmGRU.py

```
"""
Homework 8: Create GRU network
Author: Varun Aggarwal
Last Modified: 25 Apr 2022
Modifed from DLStudioV2.2.2
"""

import torch.nn as nn
import torch
class custom_pmGRU(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, batch_size=1,
       ↪ num_layers=1):
        """
        -- input_size is the size of the tensor for each word in a sequence
           ↪ of words. If you word2vec
               embedding, the value of this variable will always be equal
                  ↪ to 300.
        -- hidden_size is the size of the hidden state in the RNN
        -- output_size is the size of output of the RNN. For binary
           ↪ classification of
               input text, output_size is 2.
        -- num_layers creates a stack of GRUs
        """
        super().__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.batch_size = batch_size
        # self.gru = nn.GRU(input_size, hidden_size, num_layers)
        self.pmgru = pmGRU_mod(input_size, hidden_size, hidden_size,
           ↪ batch_size, num_layers)
        self.fc = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()
        self.logsoftmax = nn.LogSoftmax(dim=1)

    def forward(self, x, h, sequence_end=False):
        out, h = self.pmgru(x, h, sequence_end)
        # out, h = self.gru(x, h)
        out = self.fc(self.relu(out[:,-1]))
        out = self.logsoftmax(out)
        return out, h

    def init_hidden(self):
        weight = next(self.parameters()).data
        # num_layers batch_size hidden_size
        hidden = weight.new( self.num_layers, self.batch_size, self.
            ↪ hidden_size ).zero_()
        # hidden = weight.new(self.batch_size, self.hidden_size).zero_()
        return hidden

class pmGRU_mod(nn.Module):
    """
    This GRU implementation is based primarily on a "Minimal Gated" version
       ↪  of a GRU as described in
    "Simplified Minimal Gated Unit Variations for Recurrent Neural Networks
       ↪ " by Joel Heck and Fathi
    Salem. The Wikipedia page on "Gated_recurrent_unit" has a summary
       ↪ presentation of the equations
    proposed by Heck and Salem.
    """
    def __init__(self, input_size, hidden_size, output_size, batch_size,
       ↪ num_layers):
        super().__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.batch_size = batch_size
        self.num_layers = num_layers
        ## for forget gate:
        self.project1 = nn.Sequential( nn.Linear(self.input_size + self.
            ↪ hidden_size, self.hidden_size), nn.Sigmoid() )
        ## for interim out:
        self.project2 = nn.Sequential( nn.Linear( self.input_size + self.
            ↪ hidden_size, self.hidden_size), nn.Tanh() )
        ## for final out
        self.project3 = nn.Sequential( nn.Linear( self.hidden_size, self.
            ↪ output_size ), nn.Tanh() )

    def forward(self, x, h, sequence_end=False):
        combined1 = torch.cat((x, h), 2)
        forget_gate = self.project1(combined1)
        interim = forget_gate * h
        combined2 = torch.cat((x, interim), 2)
        output_interim = self.project2( combined2 )
        output = (1 - forget_gate) * h + forget_gate * output_interim
        if sequence_end == False:
            return output, output
        else:
            final_out = self.project3(output)
            return final_out, final_out

    def init_hidden(self):
        weight = next(self.parameters()).data
        hidden = weight.new(self.num_layers, self.batch_size, self.
            ↪ hidden_size).zero_()
        return hidden
```

4 Lessons Learned

The programming homework was straightforward and covered the basics of GRU implementation for sentiment analysis.

5 Suggested Enhancements

No enhancements for this homework. Well designed ! Although, the objectives of Task 1 and 2 could have been better clarified.