HW2-CSCI544

September 30, 2021

[1]:
```
import pandas as pd
import numpy as np
import scipy.sparse
import nltk
import time
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split
from sklearn import metrics

import gensim.downloader as api
from gensim.models import Word2Vec
from gensim.test.utils import datapath
from gensim import utils

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
import torch.optim as optim
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\xmh91\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

0.0.1 Quick Reload models and data, set reload to True to run accordingly, self-use

[29]:
```
reload_data = False
```

[28]:
```
if reload_data:
    ### Reloading W2V models after Kernel restart
    w2v_google_model = api.load('word2vec-google-news-300')
    model = Word2Vec.load("tained_word2vec.model")
    #model = Word2Vec.load("tained_word2vec_random1.model")
    w2v_review_model = model.wv
```

[4]:
```
if reload_data:
    ### Reload 250k Raw review data in df after Kernel restart
    df = pd.read_csv("250k_classified_reviews.csv", sep='\t')
```

[5]:
```
if reload_data:
    ## Reload 250k cleaned & preprocessed data in df After Kernel restart
    df = pd.read_csv("250k_cleaned_review_random.csv", sep='\t')
    # Drop review with nan as value
    df = df.dropna(subset=['review'])
    # Strip extra white spaces
    df['review'] = [review.strip() for review in df['review']]
    # Replace empty string with NaN
    df['review'].replace('', np.nan, inplace=True)
    # Drop review with nan as value
    df = df.dropna(subset=['review'])
```

0.1 1. Dataset Generation

[6]:
```
# Read data from csv
#raw_df = pd.read_csv("amazon_reviews_us_Kitchen_v1_00.tsv.gz", sep='\t',␣
 ↪error_bad_lines=False, warn_bad_lines=False)
raw_df = pd.read_csv("data.tsv", sep='\t', on_bad_lines="skip")
```

0.1.1 Keep Reviews and Ratings

Same as in HW1, entries with empty review content is dropped here

[7]:
```
 df = raw_df[['star_rating', 'review_body']].dropna(subset=['review_body'])
```

select 50k reviews randomly from each rating.

[8]:
```
df1=df[df['star_rating'] == 1 ].sample(n = 50000, random_state = 6)
df2=df[df['star_rating'] == 2 ].sample(n = 50000, random_state = 7)
df3=df[df['star_rating'] == 3 ].sample(n = 50000, random_state = 8)
df4=df[df['star_rating'] == 4 ].sample(n = 50000, random_state = 9)
df5=df[df['star_rating'] == 5 ].sample(n = 50000, random_state = 10)

frame = [df1, df2, df3, df4, df5]
df = pd.concat(frame)
```

0.1.2 Labelling Reviews:

We assume that ratings more than 3 denote positive 1 sentiment (class 1)

rating less than 3 denote negative sentiment (class 2).

Reviews with rating 3 are considered to have neutral sentiment (class 3).

Similar steps as HW1

[9]:
```
# Lebal samples according to star_rating
df.loc[df['star_rating'] < 3, 'star_rating'] = 2
df.loc[df['star_rating'] > 3, 'star_rating'] = 1
df.loc[df['star_rating'] == 3, 'star_rating'] = 3

df.columns = ["class", "review"]
```

0.1.3 Store 250k classified raw reviews to csv file

[10]:
```
store_data = False
```

[11]:
```
if store_data:
    df.to_csv('250k_classified_reviews.csv', sep='\t', index=False)
```

0.1.4 Reload 250k classified raw reviews from csv file

[12]:
```
if reload_data:
    df = pd.read_csv("250k_classified_reviews.csv", sep='\t')
```

0.1.5 Training data split

[13]:
```
x = df['review']
y = df['class']
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 1,␣
 ↪test_size = 0.2)
```

0.2 2. Word Embedding

download pretrained model word2vec-google-news-300

0.2.1 2(a) Experimenting with the pretrained model

[14]:
```
# Get pretrained W2V model
w2v_google_model = api.load('word2vec-google-news-300')
```

[185]:
```
# king - man + woman = queen
queen = w2v_google_model.most_similar(positive=['king', 'woman'],␣
 ↪negative=['man'])
queen
```

[185]:
```
[('queen', 0.7118193507194519),
 ('monarch', 0.6189674139022827),
 ('princess', 0.5902431011199951),
 ('crown_prince', 0.5499460697174072),
 ('prince', 0.5377321839332581),
 ('kings', 0.5236844420433044),
 ('Queen_Consort', 0.5235945582389832),
 ('queens', 0.5181134343147278),
```

```
 ('sultan', 0.5098593831062317),
 ('monarchy', 0.5087411999702454)]
```

[186]:
```
# capitalA - countryA + countryB = capitalB?
capital = w2v_google_model.most_similar(positive=['Washington', 'China'],␣
 ↪negative=['US'])
capital
```

[186]:
```
[('Beijing', 0.54819655418396),
 ('Shanghai', 0.45855337381362915),
 ('Bejing', 0.44562390446662903),
 ('Taipei', 0.44435012340545654),
 ('Beijng', 0.43815258145332336),
 ('Liaoning_Province', 0.4362034797668457),
 ('Chongqing_Evening', 0.4351152181625366),
 ('Hu', 0.4345473051071167),
 ('Hangzhou', 0.4315878748893738),
 ('Guangdong', 0.42471086978912354)]
```

0.2.2 2(b) Train a Word2Vec model

[164]:
```
# To track the training time
start_time = time.time()

reviews_tokenized = [review.split() for review in x_train.values]

model = Word2Vec(sentences=reviews_tokenized, vector_size=300, window=11,␣
 ↪min_count=10)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 60.95007944107056 seconds ---
```

0.2.3 Export trained model

[165]:
```
model.save("tained_word2vec.model")
```

0.2.4 Import self trained model if needed

[35]:
```
model = Word2Vec.load("tained_word2vec.model")
w2v_review_model = model.wv
```

[168]:
```
queen = w2v_review_model.most_similar(positive=['king', 'woman'],␣
 ↪negative=['man'])
queen
```

[168]:
```
[('queen', 0.5365573763847351),
 ('11\\\\"', 0.49044883251190186),
 ('duvet', 0.47028398513793945),
 ('13&#34;', 0.45197463035583496),
 ('9&#34;', 0.43679526448249817),
 ('Tablespoon', 0.4344377815723419),
 ('XL', 0.43140971660614014),
 ('3qt', 0.4296371340751648),
 ('saucepan,', 0.4273172914981842),
 ('sizing', 0.42034393548965454)]
```

[169]:
```
capital = w2v_review_model.most_similar(positive=['Washington', 'China'],␣
↪negative=['US'])
capital
```

[169]:
```
[('USA.<br', 0.5001760125160217),
 ('Germany', 0.4997105598449707),
 ('America.', 0.4971461594104767),
 ('China!', 0.49489426612854004),
 ('Spain', 0.4916647672653198),
 ('Germany,', 0.48961275815963745),
 ('Spain.', 0.4873398542404175),
 ('England', 0.4848261773586273),
 ('Thailand.', 0.4846789240837097),
 ('USA', 0.4741653800010681)]
```

0.2.5 What do you conclude from comparing vectors generated by yourself and the pretrained model?

The classic king - man + woman = queen example worked as intended for both models. The second example I tried was to compute the capital city by subtracting the country then plus another country, the result of pretrained model is correct again, with Beijing being the captial of China, also, there are other Chinese cities listed in the top ten results. However, for the self trained model, the top 10 results are only different countries, I think the self trained model did not ditinguish the difference between a country and a city.

Based on the king-queen and capital city result from above. I think the pretrained model has a much bigger corpus compare to Amazon reviews. Google model has 3 million vocabulary, which have more sentences from articles of political and international news and topics, while Amazon reviews are only focused on the quality of the kitchen product so maybe the relationship between city and country rarely appear in the reviews, thus it's difficult for the self-trained model to learn that the capital of "China" is "Beijing".

I think in order to get a good prediction from word2vec, the training data and testing data should share similar text context or even come from the same corpus.

0.2.6 Which of the Word2Vec models seems to encode semantic similarities between words better?

As dicussed above, the google-news model is better.

0.3 3. Simple Models

0.3.1 Cleaning and Preprocessing

0.3.2 Clean and preprocess df_sm to keep only the important word

[15]:
```
def remove_html_url(s):
    # parse html
    soup = BeautifulSoup(s, "html.parser")

    for data in soup(['style', 'script']):
        # remove tags
        data.decompose()
    # replace url with empty string and return
    return re.sub(r"http\S+", "", ' '.join(soup.stripped_strings))

# Remove HTML markups and URL in text format
#df_sm['review'] = [ remove_html_url(review) for review in df_sm['review'] ]
```

[16]:
```
contractions = {
        "a'ight": "alright",
        "ain't": "am not",
        "amn't": "am not",
        "arencha": "are not you",
        "aren't": "are not",
        "‘bout": "about",
        "cannot": "can not",
        "can't": "cannot",
        "cap’n": "captain",
        "cause": "because",
        "’cept": "except",
        "could've": "could have",
        "couldn't": "could not",
        "couldn't've": "could not have",
        "dammit": "damn it",
        "daren't": "dare not",
        "daresn't": "dare not",
        "dasn't": "dare not",
        "didn't": "did not",
        "doesn't": "does not",
        "don't": "do not",
        "dunno": "do not know",
        "d'ye": "did you",
        "e'en": "even",
```

```
        "e'er": "ever",
        "em": "them",
        "everybody's": "everybody is",
        "everyone's": "everyone is",
        "fo’c’sle": "forecastle",
        "’gainst": "against",
        "g'day": "good day",
        "gimme": "give me",
        "giv'n": "given",
        "gonna": "going to",
        "gon't": "go not",
        "gotta": "got to",
        "hadn't": "had not",
        "had've": "had have",
        "hasn't": "has not",
        "haven't": "have not",
        "he'd": "he would",
        "he'll": "he will",
        "helluva": "hell of a",
        "he's": "he is",
        "here's": "here is",
        "how'd": "how did",
        "howdy": "how do you do",
        "how'll": "how will",
        "how're": "how are",
        "how's": "how is",
        "i'd": "i would",
        "i'd've": "i would have",
        "i'll": "i will",
        "i'm": "i am",
        "imma": "i am about to",
        "i'm'o": "i am going to",
        "innit": "is it not",
        "ion": "i do not",
        "i've": "i have",
        "isn't": "is not",
        "it'd": "it would",
        "it'll": "it will",
        "it's": "it is",
        "iunno": "i do not know",
        "kinda": "kind of",
        "let's": "let us",
        "ma'am": "madam",
        "mayn't": "may not",
        "may've": "may have",
        "methinks": "i think",
        "mightn't": "might not",
```

```
        "might've": "might have",
        "mustn't": "must not",
        "mustn't've": "must not have",
        "must've": "must have",
        "‘neath": "beneath",
        "needn't": "need not",
        "nal": "and all",
        "ne'er": "never",
        "o'clock": "of the clock",
        "o'er": "over",
        "ol'": "old",
        "oughtn't": "ought not",
        "‘round": "around",
        "s": "is",
        "shalln't": "shall not",
        "shan't": "shall not",
        "she'd": "she would",
        "she'll": "she will",
        "she's": "she is",
        "should've": "should have",
        "shouldn't": "should not",
        "shouldn't've": "should not have",
        "somebody's": "somebody is",
        "someone's": "someone is",
        "something's": "something is",
        "so're": "so are",
        "so’s": "so has",
        "so’ve": "so have",
        "that'll": "that will",
        "that're": "that are",
        "that's": "that is",
        "that'd": "that would",
        "there'd": "there would",
        "there'll": "there will",
        "there're": "there are",
        "there's": "there is",
        "these're": "these are",
        "these've": "these have",
        "they'd": "they would",
        "they'll": "they will",
        "they're": "they are",
        "they've": "they have",
        "this's": "this is",
        "those're": "those are",
        "those've": "those have",
        "thout": "without",
        "’til": "until",
```

```
        "tis": "it is",
        "to've": "to have",
        "twas": "it was",
        "tween": "between",
        "twere": "it were",
        "wanna": "want to",
        "wasn't": "was not",
        "we'd": "we would",
        "we'd've": "we would have",
        "we'll": "we will",
        "we're": "we are",
        "we've": "we have",
        "weren't": "were not",
        "whatcha": "what are you",
        "what'd": "what did",
        "what'll": "what will",
        "what're": "what are/what were",
        "what's": "what is",
        "what've": "what have",
        "when's": "when is",
        "where'd": "where did",
        "where'll": "where will",
        "where're": "where are",
        "where's": "where is",
        "where've": "where have",
        "which'd": "which had",
        "which'll": "which will",
        "which're": "which are",
        "which's": "which is",
        "which've": "which have",
        "who'd": "who would",
        "who'd've": "who would have",
        "who'll": "who will",
        "who're": "who are",
        "who's": "who is",
        "who've": "who have",
        "why'd": "why did",
        "why're": "why are",
        "why's": "why is",
        "willn't": "will not",
        "won't": "will not",
        "wonnot": "will not",
        "would've": "would have",
        "wouldn't": "would not",
        "wouldn't've": "would not have",
        "y'all": "you all",
        "y'all'd've": "you all would have",
```

```
        "y'all'd'n't've": "you all would not have",
        "y'all're": "you all are",
        "y'all'ren't": "you all are not",
        "y'at": "you at",
        "yes'm": "yes madam",
        "yessir": "yes sir",
        "you'd": "you would",
        "you'll": "you will",
        "you're": "you are",
        "you've": "you have",
        "yrs": "years",
        "ur" : "your",
        "urs" : "yours"
}

def contractionfunction(s):
    for word in s.split(" "):
        if word in contractions:
            s = s.replace(word, contractions[word])
    return s

#df_sm['review'] = [ contractionfunction(review) for review in df_sm['review'] ]
```

[17]:
```
def remove_non_alphabetical(s):
    # remove single quote in word like " husband's "
    s = re.sub(r'\'', '', s)

    # replace non-alphabetical by whitespace
    s = re.sub(r"[^a-zA-Z]", ' ', s)

    # remove punctuation and return
    #return ' '.join([word.strip(string.punctuation) for word in s.split(" ")])
    return s

#df_sm['review'] = [ remove_non_alphabetical(review) for review in␣
↪df_sm['review'] ]
```

[18]:
```
#df_sm['review'] = [ re.sub(r'\s+', ' ', review) for review in df_sm['review'] ]
```

[19]:
```
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

def remove_stopwords(s):
    # only keep words not in stopwords set
    filtered_words = [word for word in s.split(" ") if word not in stop_words]
    return " ".join(filtered_words)
```

```
#df_sm['review'] = [ remove_stopwords(review) for review in df_sm['review'] ]
```

[20]:
```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()

# simple lemmatize
def easy_lemmatize(s):
    lemmatized_words = [lemmatizer.lemmatize(word) for word in s.split(" ")]
    return " ".join(lemmatized_words)

# use simple lemmatize
#df_sm['review'] = [ easy_lemmatize(review) for review in df_sm['review'] ]
```

0.3.3 Cleanning the 250k raw data

[21]:
```
# Keep only class=1 and class =2 data
#df_sm = pd.concat([df[df['class'] == 1], df[df['class'] == 2]])


def data_clean(x):
    x = x.str.lower()
    x = [ remove_html_url(review) for review in x ]
    x = [ contractionfunction(review) for review in x ]
    x = [ remove_non_alphabetical(review) for review in x ]
    x = [ re.sub(r'\s+', ' ', review) for review in x ]
    x = [ remove_stopwords(review) for review in x ]
    x = [ easy_lemmatize(review) for review in x ]
    x = [ review.strip() for review in x]
    return x

df_copy = df.copy()

df["review"] = data_clean(df["review"])
df.replace('', np.nan, inplace=True)
```

```
C:\Users\xmh91\.conda\envs\py39-CS544\lib\site-packages\bs4\__init__.py:431:
MarkupResemblesLocatorWarning: "http://www.consumerreports.org/cro/magazine-
archive/2011/november/appliances/can-you-stop-stirring/overview/index.htm" looks
like a URL. Beautiful Soup is not an HTTP client. You should probably use an
HTTP client like requests to get the document behind the URL, and feed that
document to Beautiful Soup.
  warnings.warn(
C:\Users\xmh91\.conda\envs\py39-CS544\lib\site-packages\bs4\__init__.py:431:
MarkupResemblesLocatorWarning: "https://www.facebook.com/cherischocolates" looks
like a URL. Beautiful Soup is not an HTTP client. You should probably use an
```

```
HTTP client like requests to get the document behind the URL, and feed that
document to Beautiful Soup.
  warnings.warn(
```

0.3.4 Store cleaned and preprocessed data into file

[37]:
```
if store_data:
    df.to_csv('250k_cleaned_review_random.csv', sep='\t', index=False)
```

0.3.5 Reload df_sm if needed

[39]:
```
#df_sm = pd.read_csv("200k_cleaned_review_binary_classes.csv", sep='\t')
if reload_data:
    df = pd.read_csv("250k_cleaned_review_random.csv", sep='\t')
```

[22]:
```
# Drop review with nan as value
df = df.dropna(subset=['review'])
# Strip extra white spaces
df['review'] = [review.strip() for review in df['review']]
# Replace empty string with NaN
df['review'].replace('', np.nan, inplace=True)
# Drop review with nan as value
df = df.dropna(subset=['review'])
```

[23]:
```
# Data for binary classification
df_sm = pd.concat([df[df['class'] == 1], df[df['class'] == 2]])

df_sm.shape
```

[23]:
```
(199884, 2)
```

0.3.6 Split train/test data again

[24]:
```
df_sm = df_sm.dropna(subset=['review'])
# Split train and test data
x_sm = df_sm["review"]
y_sm = df_sm["class"]

x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x_sm, y_sm,␣
 ↪random_state = 12, test_size = 0.2)
```

0.3.7 Create document vector for each review based on the average vector of each word in each review

0.3.8 Preparing Vectors for Model Training

[25]:
```
# Debug DocVec Function
def computeDocVec(w2v_model, train_input):
    docVec = []

    for review in train_input:
        rev_vec = []
        for word in review.split():
            # add word embedding to rev_vec if not in W2V, add zeros
            if word in w2v_model:
                rev_vec.append(w2v_model[word])
            else:
                rev_vec.append(np.zeros(300,))
        docVec.append(np.mean(rev_vec, axis=0))
    return docVec
```

[30]:
```
# Prepare Vectors
review_train_vec = scipy.sparse.csr_matrix(np.
 ↪array(computeDocVec(w2v_review_model, x_train_sm), dtype="float64"))
review_test_vec = scipy.sparse.csr_matrix(np.
 ↪array(computeDocVec(w2v_review_model, x_test_sm), dtype="float64"))
```

[31]:
```
# Prepare Vectors
google_train_vec = scipy.sparse.csr_matrix(np.
 ↪array(computeDocVec(w2v_google_model, x_train_sm), dtype="float64"))
google_test_vec = scipy.sparse.csr_matrix(np.
 ↪array(computeDocVec(w2v_google_model, x_test_sm), dtype="float64"))
```

[32]:
```
# training data output
y_train_sm = y_train_sm.astype('int')
y_test_sm = y_test_sm.astype('int')
```

0.4 Perceptron

[33]:
```
from sklearn.linear_model import Perceptron

# set iteration and learning rate
max_iter = 1000
eta0 = 0.1
```

0.4.1 Model #1. Self-trained W2V Perceptron

[34]:
```
# Train the model with train dataset and train targets
ppn_self_clf = Perceptron()
ppn_self_clf.fit(review_train_vec, y_train_sm)
self_y_pred = ppn_self_clf.predict(review_test_vec)
print("Perceptron Self-trained W2V Accuracy:")
print(metrics.accuracy_score(y_test_sm, self_y_pred))
```

```
Perceptron Self-trained W2V Accuracy:
0.8076894214173149
```

0.4.2 Model #2. Pre-trained W2V Perceptron

[35]:
```
# Train the model with train dataset and train targets
ppn_google_clf = Perceptron()
ppn_google_clf.fit(google_train_vec, y_train_sm)
google_y_pred = ppn_google_clf.predict(review_test_vec)
print("Perceptron Pretrained W2V Accuracy:")
print(metrics.accuracy_score(y_test_sm, google_y_pred))
```

```
Perceptron Pretrained W2V Accuracy:
0.5128448858093404
```

0.4.3 Perception TF-IDF Vectors from HW1

Accuracy

0.89514375

0.5 SVM

[36]:
```
from sklearn.svm import LinearSVC

# Train the model with train dataset and train targets
svm = LinearSVC(random_state=0, tol=1e-5, max_iter=2000)
```

0.5.1 Model #3. Self-trained W2V SVM

[37]:
```
# To track the training time
start_time = time.time()

# Train the model
svm.fit(review_train_vec, y_train_sm)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 217.10737490653992 seconds ---
```

```
C:\Users\xmh91\.conda\envs\py39-CS544\lib\site-
packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

[38]:
```
# predict on train dataset
y_pred = svm.predict(review_test_vec)
```

[39]:
```
print("SVM Self-trained W2V Accuracy:")
print(metrics.accuracy_score(y_test_sm, y_pred))
```

```
SVM Self-trained W2V Accuracy:
0.8467368737023788
```

0.5.2 Model #4. Pre-trained W2V SVM

[40]:
```
svm = LinearSVC(random_state=0, tol=1e-5, max_iter=2000)
```

[41]:
```
# To track the training time
start_time = time.time()

# Train the model
svm.fit(google_train_vec, y_train_sm)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 41.98566770553589 seconds ---
```

[42]:
```
# predict on train dataset
y_pred = svm.predict(review_test_vec)
```

[43]:
```
print("SVM Pretrained W2V Accuracy:")
print(metrics.accuracy_score(y_test_sm, y_pred))
```

```
SVM Pretrained W2V Accuracy:
0.533331665707782
```

Conclusion: The perceptron and SVM trained on the vectors generated by the self-trained Word2Vec model could accurately (80%~85%) predict the sentiment of the review, while the two models trained on vectors generated by pre-trained Word2Vec model only have around 50% accuracy, basically like coin-flip guess. I think the self-trained W2V model performs better because the entire corpus contains only reviews, while the pre-trained W2V model were trained on news articles.

0.6 Feedforward Neural Netword

****Note, you might have to restart the kernel due to GPU memory usage being high after each model training**

0.6.1 Model #5. Self-trained W2V FNN-Binary - DocVec

[44]:
```
class ThreeLayerMLP(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.fc1 = torch.nn.Linear(input_size, 50)
        self.fc2 = torch.nn.Linear(50, 10)

    def forward(self, x):
        hidden = F.relu(self.fc1(x))
        out = self.fc2(hidden)
        return out

# Calculate Accuracy
def getAccuracy(out, labels):
    _, predict = torch.max(out.data, 1)
    total = labels.shape[0]*1.0
    correct = (labels == predict).sum().item()
    return correct/total
```

[46]:
```
# Prepare Tensors
# Split train and test
x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x_sm, y_sm,␣
 ↪test_size = 0.2)

features_train = np.array(computeDocVec(w2v_review_model, x_train_sm),␣
 ↪dtype="float64")
features_test = np.array(computeDocVec(w2v_review_model, x_test_sm),␣
 ↪dtype="float64")

X_train,X_test,Y_train,Y_test=torch.from_numpy(features_train).float(),torch.
 ↪from_numpy(features_test).float(),torch.from_numpy(y_train_sm.to_numpy()).
 ↪type(torch.LongTensor),torch.from_numpy(y_test_sm.to_numpy()).type(torch.
 ↪LongTensor)

# Update lables to be 0 and 1
Y_train = Y_train - 1
Y_test = Y_test - 1
```

Train Model

[47]:
```
print("Binary classification nn - self trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Self W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
```

```
    print("Self+ W2V Model trainning on CPU...")

# Initiate model
model = ThreeLayerMLP(300)
model.to(device)

# Configure Hyperparamters
loss_fn = torch.nn.CrossEntropyLoss()
lr = 0.00005
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 5000
batch = 10000

X_train_cuda = X_train.to(device)
X_test_cuda = X_test.to(device)
Y_train_cuda = Y_train.to(device)
Y_test_cuda = Y_test.to(device)

for e in range(1,epochs+1):
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = loss_fn(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
    # Update weights and bias
    optimizer.step()

    # Report Accuracy each 100 epochs
    if e % 500 == 0:
        test_out = model(X_test_cuda)
        valid_loss = loss_fn(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
Binary classification nn - self trained W2V
Self W2V Model trainning on GPU…
Epoch: 500 train_loss: 0.6033 valid_loss: 0.6017 accuracy: 0.7997
Epoch: 1000 train_loss: 0.4387 valid_loss: 0.4384 accuracy: 0.8244
Epoch: 1500 train_loss: 0.3989 valid_loss: 0.3991 accuracy: 0.8341
Epoch: 2000 train_loss: 0.3811 valid_loss: 0.3817 accuracy: 0.8400
Epoch: 2500 train_loss: 0.3707 valid_loss: 0.3715 accuracy: 0.8433
Epoch: 3000 train_loss: 0.3629 valid_loss: 0.3642 accuracy: 0.8463
Epoch: 3500 train_loss: 0.3558 valid_loss: 0.3577 accuracy: 0.8492
Epoch: 4000 train_loss: 0.3488 valid_loss: 0.3515 accuracy: 0.8512
Epoch: 4500 train_loss: 0.3421 valid_loss: 0.3459 accuracy: 0.8529
Epoch: 5000 train_loss: 0.3361 valid_loss: 0.3414 accuracy: 0.8531
```

[ ]:
```
# Clear GPU Cache After Training
del X_train_cuda
del X_test_cuda
del Y_train_cuda
del Y_test_cuda
del model
del y_pred
del loss
del optimizer
torch.cuda.empty_cache()
print(torch.cuda.memory_allocated())
print(torch.cuda.memory_reserved())
```

0.6.2 Model #6. Pre-trained W2V FNN-Binary - DocVec

[20]:
```
# Split train and test
x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x_sm, y_sm,␣
 ↪test_size = 0.2)

features_train = np.array(computeDocVec(w2v_google_model, x_train_sm),␣
 ↪dtype="float64")
features_test = np.array(computeDocVec(w2v_google_model, x_test_sm),␣
 ↪dtype="float64")

X_train,X_test,Y_train,Y_test=torch.from_numpy(features_train).float(),torch.
 ↪from_numpy(features_test).float(),torch.from_numpy(y_train_sm.to_numpy()).
 ↪type(torch.LongTensor),torch.from_numpy(y_test_sm.to_numpy()).type(torch.
 ↪LongTensor)

# Update lables to be 0 and 1
Y_train = Y_train - 1
Y_test = Y_test - 1
```

Train Model

[21]:
```
print("Binary classification nn - Google pre-trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Google W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
    print("Google W2V Model trainning on CPU...")

# Initiate model
model = ThreeLayerMLP(300)
model.to(device)
```

```
# Configure Hyperparamters
loss_fn = torch.nn.CrossEntropyLoss()
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 5000
batch = 10000

X_train_cuda = X_train.to(device)
X_test_cuda = X_test.to(device)
Y_train_cuda = Y_train.to(device)
Y_test_cuda = Y_test.to(device)

for e in range(1,epochs+1):
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = loss_fn(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
    # Update weights and bias
    optimizer.step()

    # Report Accuracy each 100 epochs
    if e % 500 == 0:
        test_out = model(X_test_cuda)
        valid_loss = loss_fn(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
Binary classification nn - Google pre-trained W2V
Google W2V Model trainning on GPU…
Epoch: 500 train_loss: 0.7796 valid_loss: 0.7777 accuracy: 0.6412
Epoch: 1000 train_loss: 0.5728 valid_loss: 0.5708 accuracy: 0.7770
Epoch: 1500 train_loss: 0.4959 valid_loss: 0.4941 accuracy: 0.7880
Epoch: 2000 train_loss: 0.4581 valid_loss: 0.4570 accuracy: 0.8000
Epoch: 2500 train_loss: 0.4360 valid_loss: 0.4358 accuracy: 0.8090
Epoch: 3000 train_loss: 0.4222 valid_loss: 0.4228 accuracy: 0.8155
Epoch: 3500 train_loss: 0.4130 valid_loss: 0.4142 accuracy: 0.8197
Epoch: 4000 train_loss: 0.4054 valid_loss: 0.4072 accuracy: 0.8223
Epoch: 4500 train_loss: 0.3984 valid_loss: 0.4008 accuracy: 0.8249
Epoch: 5000 train_loss: 0.3924 valid_loss: 0.3953 accuracy: 0.8269
```

[8]:
```
def clearCache():
    # Clear GPU Cache After Training
    del X_train_cuda
    del X_test_cuda
```

```
    del Y_train_cuda
    del Y_test_cuda
    torch.save(model,"pre_trained_nn_model_binary.pkl")
    del model
    del y_pred
    del loss
    del optimizer
    torch.cuda.empty_cache()
    print(torch.cuda.memory_allocated())
    print(torch.cuda.memory_reserved())
```

0.6.3 Model #7. Self-trained W2V FNN-Ternary - DocVec

[25]:
```
x_tri = df["review"]
y_tri = df["class"]

x_train_tri, x_test_tri, y_train_tri, y_test_tri = train_test_split(x_tri,␣
↪y_tri, random_state = 11, test_size = 0.2)

features_train_tri = np.array(computeDocVec(w2v_review_model, x_train_tri),␣
↪dtype="float64")
features_test_tri = np.array(computeDocVec(w2v_review_model, x_test_tri),␣
↪dtype="float64")

X_train_tri,X_test_tri,Y_train_tri,Y_test_tri=torch.
↪from_numpy(features_train_tri).float(),torch.from_numpy(features_test_tri).
↪float(),torch.from_numpy(y_train_tri.to_numpy()).type(torch.
↪LongTensor),torch.from_numpy(y_test_tri.to_numpy()).type(torch.LongTensor)

# Update lables to be 0 and 1
Y_train_tri = Y_train_tri - 1
Y_test_tri = Y_test_tri - 1
```

[28]:
```
print("ternary classification nn - self trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Self W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
    print("Self W2V Model trainning on CPU...")

# Initiate model
model = TriMLP(300)
model.to(device)

# Configure Hyperparamters
```

```
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 400
batch = 2000

X_train_cuda = X_train_tri.to(device)
X_test_cuda = X_test_tri.to(device)
Y_train_cuda = Y_train_tri.to(device)
Y_test_cuda = Y_test_tri.to(device)

for e in range(1,epochs+1):
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = F.nll_loss(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
    # Update weights and bias
    optimizer.step()

    if e % 40 == 0:
        test_out = model(X_test_cuda)
        valid_loss = F.nll_loss(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
 ↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
ternary classification nn - self trained W2V
Self W2V Model trainning on GPU…
Epoch: 40 train_loss: 1.0155 valid_loss: 1.0160 accuracy: 0.5659
Epoch: 80 train_loss: 0.9640 valid_loss: 0.9655 accuracy: 0.6126
Epoch: 120 train_loss: 0.9179 valid_loss: 0.9206 accuracy: 0.6293
Epoch: 160 train_loss: 0.8805 valid_loss: 0.8842 accuracy: 0.6384
Epoch: 200 train_loss: 0.8503 valid_loss: 0.8546 accuracy: 0.6455
Epoch: 240 train_loss: 0.8261 valid_loss: 0.8309 accuracy: 0.6520
Epoch: 280 train_loss: 0.8067 valid_loss: 0.8119 accuracy: 0.6573
Epoch: 320 train_loss: 0.7906 valid_loss: 0.7962 accuracy: 0.6626
Epoch: 360 train_loss: 0.7786 valid_loss: 0.7845 accuracy: 0.6663
Epoch: 400 train_loss: 0.7697 valid_loss: 0.7758 accuracy: 0.6694
```

0.6.4 Model #8. Pre-trained W2V FNN-Ternary - DocVec

[30]:
```
x_tri = df["review"]
y_tri = df["class"]

x_train_tri, x_test_tri, y_train_tri, y_test_tri = train_test_split(x_tri,␣
 ↪y_tri, random_state = 11, test_size = 0.2)
```

```
features_train_tri = np.array(computeDocVec(w2v_google_model, x_train_tri),␣
↪dtype="float64")
features_test_tri = np.array(computeDocVec(w2v_google_model, x_test_tri),␣
↪dtype="float64")

X_train_tri,X_test_tri,Y_train_tri,Y_test_tri=torch.
↪from_numpy(features_train_tri).float(),torch.from_numpy(features_test_tri).
↪float(),torch.from_numpy(y_train_tri.to_numpy()).type(torch.
↪LongTensor),torch.from_numpy(y_test_tri.to_numpy()).type(torch.LongTensor)

# Update lables to be 0 and 1
Y_train_tri = Y_train_tri - 1
Y_test_tri = Y_test_tri - 1
```

[31]:
```
print("ternary classification nn - pre trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Self W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
    print("Self W2V Model trainning on CPU...")

# Initiate model
model = TriMLP(300)
model.to(device)

# Configure Hyperparamters
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 400
batch = 2000

X_train_cuda = X_train_tri.to(device)
X_test_cuda = X_test_tri.to(device)
Y_train_cuda = Y_train_tri.to(device)
Y_test_cuda = Y_test_tri.to(device)

for e in range(1,epochs+1):
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = F.nll_loss(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
```

```
    # Update weights and bias
    optimizer.step()

    if e % 40 == 0:
        test_out = model(X_test_cuda)
        valid_loss = F.nll_loss(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
ternary classification nn - pre trained W2V
Self W2V Model trainning on GPU…
Epoch: 40 train_loss: 1.1040 valid_loss: 1.1038 accuracy: 0.2012
Epoch: 80 train_loss: 1.0887 valid_loss: 1.0884 accuracy: 0.4942
Epoch: 120 train_loss: 1.0717 valid_loss: 1.0713 accuracy: 0.4202
Epoch: 160 train_loss: 1.0545 valid_loss: 1.0542 accuracy: 0.4080
Epoch: 200 train_loss: 1.0423 valid_loss: 1.0421 accuracy: 0.4172
Epoch: 240 train_loss: 1.0318 valid_loss: 1.0318 accuracy: 0.4746
Epoch: 280 train_loss: 1.0212 valid_loss: 1.0213 accuracy: 0.5461
Epoch: 320 train_loss: 1.0096 valid_loss: 1.0097 accuracy: 0.5883
Epoch: 360 train_loss: 0.9964 valid_loss: 0.9965 accuracy: 0.6070
Epoch: 400 train_loss: 0.9823 valid_loss: 0.9824 accuracy: 0.6160
```

0.6.5 Concatenate 10 Word Vectors

[4]:
```
class TriMLP(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.fc1 = torch.nn.Linear(input_size, 50)
        self.fc2 = torch.nn.Linear(50, 10)
        self.fc3 = torch.nn.Linear(10, 3)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return F.log_softmax(self.fc3(x), dim=1)

# Calculate Accuracy
def getAccuracy(out, labels):
    _, predict = torch.max(out.data, 1)
    total = labels.shape[0]*1.0
    correct = (labels == predict).sum().item()
    return correct/total
```

[10]:
```
# Concatenate 10 vec of each review
def concatDocVec(w2v_model, train_input):
    res = []

    for review in train_input:
```

```
        words = review.split()
        doc_vec = []
        if len(words) >= 10:
            for w in words[:10]:
                if w in w2v_model:
                    doc_vec = np.concatenate((doc_vec, w2v_model[w]), axis=None)
                else:
                    doc_vec = np.concatenate((doc_vec, np.zeros(300,)),␣
 ↪axis=None)
        else:
            for w in words:
                if w in w2v_model:
                    doc_vec = np.concatenate((doc_vec, w2v_model[w]), axis=None)
                else:
                    doc_vec = np.concatenate((doc_vec, np.zeros(300,)),␣
 ↪axis=None)
            for i in range(10 - len(words)):
                doc_vec = np.concatenate((doc_vec, np.zeros(300,)), axis=None)
        res.append(doc_vec)

    return res
```

[6]:
```
# Prepare Tensor - self Trained
x_tri = df["review"]
y_tri = df["class"]

x_train_tri, x_test_tri, y_train_tri, y_test_tri = train_test_split(x_tri,␣
 ↪y_tri, random_state = 11, test_size = 0.2)

features_train_tri = np.array(concatDocVec(w2v_review_model, x_train_tri),␣
 ↪dtype="float64")
features_test_tri = np.array(concatDocVec(w2v_review_model, x_test_tri),␣
 ↪dtype="float64")

X_train_tri,X_test_tri,Y_train_tri,Y_test_tri=torch.
 ↪from_numpy(features_train_tri).float(),torch.from_numpy(features_test_tri).
 ↪float(),torch.from_numpy(y_train_tri.to_numpy()).type(torch.
 ↪LongTensor),torch.from_numpy(y_test_tri.to_numpy()).type(torch.LongTensor)

# Update lables to be 0 and 1
Y_train_tri = Y_train_tri - 1
Y_test_tri = Y_test_tri - 1
```

0.6.6 Model #9. Self-trained W2V FNN-Ternary - ConcatVec

[9]:
```
print("ternary classification nn - self trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Self W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
    print("Self W2V Model trainning on CPU...")

# Initiate model
model = TriMLP(3000)
model.to(device)

# Configure Hyperparamters
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 400
batch = 2000

X_train_cuda = X_train_tri.to(device)
X_test_cuda = X_test_tri.to(device)
Y_train_cuda = Y_train_tri.to(device)
Y_test_cuda = Y_test_tri.to(device)

for e in range(1,epochs+1):
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = F.nll_loss(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
    # Update weights and bias
    optimizer.step()

    if e % 40 == 0:
        test_out = model(X_test_cuda)
        valid_loss = F.nll_loss(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
ternary classification nn - self trained W2V
Self W2V Model trainning on GPU…
Epoch: 40 train_loss: 0.9053 valid_loss: 0.9075 accuracy: 0.5994
Epoch: 80 train_loss: 0.8523 valid_loss: 0.8610 accuracy: 0.6200
Epoch: 120 train_loss: 0.8289 valid_loss: 0.8438 accuracy: 0.6283
```

```
Epoch: 160 train_loss: 0.8143 valid_loss: 0.8349 accuracy: 0.6311
Epoch: 200 train_loss: 0.8026 valid_loss: 0.8290 accuracy: 0.6343
Epoch: 240 train_loss: 0.7918 valid_loss: 0.8246 accuracy: 0.6363
Epoch: 280 train_loss: 0.7812 valid_loss: 0.8213 accuracy: 0.6388
Epoch: 320 train_loss: 0.7706 valid_loss: 0.8188 accuracy: 0.6398
Epoch: 360 train_loss: 0.7596 valid_loss: 0.8171 accuracy: 0.6413
Epoch: 400 train_loss: 0.7482 valid_loss: 0.8164 accuracy: 0.6420
```

0.6.7 Model #10. Pre-trained W2V FNN-Ternary - ConcatVec

[5]:
```
x_tri = df["review"]
y_tri = df["class"]

x_train_tri, x_test_tri, y_train_tri, y_test_tri = train_test_split(x_tri,␣
 ↪y_tri, random_state = 11, test_size = 0.2)

features_train_tri = np.array(concatDocVec(w2v_google_model, x_train_tri),␣
 ↪dtype="float64")
features_test_tri = np.array(concatDocVec(w2v_google_model, x_test_tri),␣
 ↪dtype="float64")

X_train_tri,X_test_tri,Y_train_tri,Y_test_tri=torch.
 ↪from_numpy(features_train_tri).float(),torch.from_numpy(features_test_tri).
 ↪float(),torch.from_numpy(y_train_tri.to_numpy()).type(torch.
 ↪LongTensor),torch.from_numpy(y_test_tri.to_numpy()).type(torch.LongTensor)

# Update lables to be 0 and 1
Y_train_tri = Y_train_tri - 1
Y_test_tri = Y_test_tri - 1
```

Train Model

[6]:
```
print("ternary classification nn - pre-trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Google W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
    print("Google W2V Model trainning on CPU...")

# Initiate model
model = TriMLP()
model.to(device)
```

```
ternary classification nn - pre-trained W2V
Google W2V Model trainning on GPU…
```

[6]:
```
TriMLP(
  (fc1): Linear(in_features=3000, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
  (fc3): Linear(in_features=10, out_features=3, bias=True)
)
```

[7]:
```
# Configure Hyperparamters
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 800

batch = 1000
```

[8]:
```
X_train_cuda = X_train_tri.to(device)
X_test_cuda = X_test_tri.to(device)
Y_train_cuda = Y_train_tri.to(device)
Y_test_cuda = Y_test_tri.to(device)
```

[9]:
```
for e in range(1,epochs+1):
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = F.nll_loss(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
    # Update weights and bias
    optimizer.step()

    if e % 80 == 0:
        test_out = model(X_test_cuda)
        valid_loss = F.nll_loss(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
Epoch: 80 train_loss: 1.0165 valid_loss: 1.0180 accuracy: 0.5670
Epoch: 160 train_loss: 0.9523 valid_loss: 0.9550 accuracy: 0.5892
Epoch: 240 train_loss: 0.9083 valid_loss: 0.9127 accuracy: 0.6002
Epoch: 320 train_loss: 0.8815 valid_loss: 0.8887 accuracy: 0.6078
Epoch: 400 train_loss: 0.8655 valid_loss: 0.8759 accuracy: 0.6122
Epoch: 480 train_loss: 0.8545 valid_loss: 0.8682 accuracy: 0.6154
Epoch: 560 train_loss: 0.8455 valid_loss: 0.8626 accuracy: 0.6173
Epoch: 640 train_loss: 0.8376 valid_loss: 0.8582 accuracy: 0.6189
Epoch: 720 train_loss: 0.8298 valid_loss: 0.8544 accuracy: 0.6201
Epoch: 800 train_loss: 0.8223 valid_loss: 0.8514 accuracy: 0.6221
```

0.6.8 Model #11. Self-trained W2V FNN-Binary - ConcatVec

[11]:
```
df_sm = pd.concat([df[df['class'] == 1], df[df['class'] == 2]])
# Split train and test
x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x_sm, y_sm,␣
 ↪test_size = 0.2)

features_train = np.array(concatDocVec(w2v_review_model, x_train_sm),␣
 ↪dtype="float64")
features_test = np.array(concatDocVec(w2v_review_model, x_test_sm),␣
 ↪dtype="float64")

X_train,X_test,Y_train,Y_test=torch.from_numpy(features_train).float(),torch.
 ↪from_numpy(features_test).float(),torch.from_numpy(y_train_sm.to_numpy()).
 ↪type(torch.LongTensor),torch.from_numpy(y_test_sm.to_numpy()).type(torch.
 ↪LongTensor)

# Update lables to be 0 and 1
Y_train = Y_train - 1
Y_test = Y_test - 1
```

[13]:
```
print("Binary classification nn - Self pre-trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Google W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
    print("Google W2V Model trainning on CPU...")

# Initiate model with input size 3000
model = ThreeLayerMLP(3000)
model.to(device)

# Configure Hyperparamters
loss_fn = torch.nn.CrossEntropyLoss()
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 1000
batch = 10000

X_train_cuda = X_train.to(device)
X_test_cuda = X_test.to(device)
Y_train_cuda = Y_train.to(device)
Y_test_cuda = Y_test.to(device)

for e in range(1,epochs+1):
```

```
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = loss_fn(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
    # Update weights and bias
    optimizer.step()

    # Report Accuracy each 100 epochs
    if e % 100 == 0:
        test_out = model(X_test_cuda)
        valid_loss = loss_fn(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
Binary classification nn - Self pre-trained W2V
Google W2V Model trainning on GPU…
Epoch: 100 train_loss: 0.5647 valid_loss: 0.5699 accuracy: 0.7644
Epoch: 200 train_loss: 0.4921 valid_loss: 0.5039 accuracy: 0.7773
Epoch: 300 train_loss: 0.4645 valid_loss: 0.4816 accuracy: 0.7820
Epoch: 400 train_loss: 0.4473 valid_loss: 0.4695 accuracy: 0.7854
Epoch: 500 train_loss: 0.4336 valid_loss: 0.4616 accuracy: 0.7881
Epoch: 600 train_loss: 0.4215 valid_loss: 0.4562 accuracy: 0.7898
Epoch: 700 train_loss: 0.4096 valid_loss: 0.4525 accuracy: 0.7906
Epoch: 800 train_loss: 0.3976 valid_loss: 0.4502 accuracy: 0.7920
Epoch: 900 train_loss: 0.3853 valid_loss: 0.4493 accuracy: 0.7930
Epoch: 1000 train_loss: 0.3728 valid_loss: 0.4499 accuracy: 0.7930
```

0.6.9 Model #12. Pre-trained W2V FNN-Binary - ConcatVec

[16]:
```
# Split train and test
x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x_sm, y_sm,␣
↪test_size = 0.2)

features_train = np.array(concatDocVec(w2v_google_model, x_train_sm),␣
↪dtype="float64")
features_test = np.array(concatDocVec(w2v_google_model, x_test_sm),␣
↪dtype="float64")

X_train,X_test,Y_train,Y_test=torch.from_numpy(features_train).float(),torch.
↪from_numpy(features_test).float(),torch.from_numpy(y_train_sm.to_numpy()).
↪type(torch.LongTensor),torch.from_numpy(y_test_sm.to_numpy()).type(torch.
↪LongTensor)

# Update lables to be 0 and 1
```

```
Y_train = Y_train - 1
Y_test = Y_test - 1
```

[17]:
```
print("Binary classification nn - pre-trained W2V")
# Set device
if torch.cuda.is_available():
    device = torch.device("cuda:0")
    print("Google W2V Model trainning on GPU...")
else:
    device = torch.device("cpu")
    print("Google W2V Model trainning on CPU...")

# Initiate model with input size 3000
model = ThreeLayerMLP(3000)
model.to(device)

# Configure Hyperparamters
loss_fn = torch.nn.CrossEntropyLoss()
lr = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 1000
batch = 10000

X_train_cuda = X_train.to(device)
X_test_cuda = X_test.to(device)
Y_train_cuda = Y_train.to(device)
Y_test_cuda = Y_test.to(device)

for e in range(1,epochs+1):
    y_pred = model(X_train_cuda)

    # Compute loss
    loss = loss_fn(y_pred, Y_train_cuda)
    optimizer.zero_grad()
    # Compute gradient
    loss.backward()
    # Update weights and bias
    optimizer.step()

    # Report Accuracy each 100 epochs
    if e % 100 == 0:
        test_out = model(X_test_cuda)
        valid_loss = loss_fn(test_out, Y_test_cuda)
        print("Epoch: {} train_loss: {:.4f} valid_loss: {:.4f} accuracy: {:.
↪4f}".format(e, loss, valid_loss, getAccuracy(test_out, Y_test_cuda)))
```

```
Binary classification nn - pre-trained W2V
Google W2V Model trainning on GPU…
```

```
Epoch: 100 train_loss: 1.1297 valid_loss: 1.1225 accuracy: 0.7143
Epoch: 200 train_loss: 0.7204 valid_loss: 0.7212 accuracy: 0.7327
Epoch: 300 train_loss: 0.6112 valid_loss: 0.6149 accuracy: 0.7430
Epoch: 400 train_loss: 0.5618 valid_loss: 0.5677 accuracy: 0.7511
Epoch: 500 train_loss: 0.5334 valid_loss: 0.5415 accuracy: 0.7563
Epoch: 600 train_loss: 0.5146 valid_loss: 0.5248 accuracy: 0.7610
Epoch: 700 train_loss: 0.5007 valid_loss: 0.5130 accuracy: 0.7635
Epoch: 800 train_loss: 0.4897 valid_loss: 0.5041 accuracy: 0.7653
Epoch: 900 train_loss: 0.4805 valid_loss: 0.4970 accuracy: 0.7665
Epoch: 1000 train_loss: 0.4725 valid_loss: 0.4914 accuracy: 0.7684
```

Conclusion: The accuracy for self-trained FNN model is a little bit better (~85%) compare to perceptron and SVM, and the accuracy for pre-trained FNN model is much better than perceptron and SVM. I think the mean reason is that in the neural networks could better classify dataset than the linear models because of the hidden layers. It could use "multiple lines" to divide the dataset rather than one line. Note: Due to the high GPU memory usage of my models, I had to restart the kernel for between running model 11 and 12

0.7 5. Recurrent Neural Networks

[2]:
```python
# create 50 vec for each review
def paddedVec(w2v_model, train_input):
    res = []

    for review in train_input:
        docVec = []
        words = review.split()
        if len(words) >= 50 :
            words = words[:50]
        else:
            #words = ["0" for _ in range(50 - len(words))] + words
            docVec = [np.zeros(300,) for _ in range(50 - len(words))]

        for word in words:
            if word in w2v_model:
                docVec.append(w2v_model[word])
            else:
                docVec.append(np.zeros(300,))

        res.append(docVec)

    return np.array(res, dtype="float64")
```

[7]:
```python
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
```

```
    device = torch.device("cpu")

device = torch.device("cpu")

class SeqRNN(nn.Module):

    def __init__(self,vocab_size,hidden_size,output_size):
        super(SeqRNN,self).__init__()
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.rnn = nn.RNN(self.vocab_size,self.hidden_size,batch_first=True)
        self.linear = nn.Linear(self.hidden_size,self.output_size)

    def forward(self,input):
        batch_size = len(input)
        h0 = torch.zeros(1,batch_size,self.hidden_size).to(device)
        output , hidden = self.rnn(input,h0)
        output = output[ : ,-1, : ]
        output = self.linear(output)
        output = torch.nn.functional.softmax(output,dim=1)
        return output
```

[8]:
```
# Utilities
###### For dataloader enumerate loop ######

def test_accuracy_loader(model, test_loader):
    total = 0
    correct = 0
    for _, (review, label) in enumerate(test_loader):
        total += len(label)
        pred = predict_review_tensor(review, model)
        correct += (pred == label).sum()
    return correct.item() * 1.0 / total

def predict_review_tensor(review, model):
    with torch.no_grad():
        out = model(review)
        out = torch.argmax(out).item()
    return out
```

0.8 5(a)

0.8.1 Model #13. Self-trained W2V RNN-Simple-Binary - PaddedVec

[14]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep only class 1 and 2 for Binary Classification
df_sm = pd.concat([df[df['class'] == 1], df[df['class'] == 2]])
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])

# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
 ↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
train_set = paddedVec(w2v_review_model, train_set)
test_set = paddedVec(w2v_review_model, test_set)

## Substract 1 from labels, so the classes starts from 0
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)


## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

Train RNN

[15]:
```
model = SeqRNN(300,50,2)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0001)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):

        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
 ↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()


torch.cuda.empty_cache()
```

```
Epoch: 0 Timetaken: 36.13s train_loss: 1.2467 accuracy: 0.7535
Epoch: 1 Timetaken: 34.67s train_loss: 0.3255 accuracy: 0.7652
Epoch: 2 Timetaken: 36.84s train_loss: 0.3242 accuracy: 0.7688
Epoch: 3 Timetaken: 35.96s train_loss: 0.3230 accuracy: 0.7430
Epoch: 4 Timetaken: 36.57s train_loss: 1.3056 accuracy: 0.7800
Epoch: 5 Timetaken: 36.72s train_loss: 0.3140 accuracy: 0.7825
Epoch: 6 Timetaken: 37.22s train_loss: 1.2956 accuracy: 0.7843
Epoch: 7 Timetaken: 36.23s train_loss: 0.3150 accuracy: 0.7827
Epoch: 8 Timetaken: 36.84s train_loss: 0.3151 accuracy: 0.7748
Epoch: 9 Timetaken: 36.62s train_loss: 0.3136 accuracy: 0.7652
```

[18]:
```
# Clear GPU Cache After Training
del train_set
del test_set
del train_labels
del test_labels
del training_set
del testing_set
```

```
del train_loader
del test_loader

del model
del loss_func
del optimizer
torch.cuda.empty_cache()
print(torch.cuda.memory_allocated())
print(torch.cuda.memory_reserved())
```

```
205824
2097152
```

0.8.2 Model #14. Pre-trained W2V RNN-Simple-Binary - PaddedVec

[19]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep only class 1 and 2 for Binary Classification
df_sm = pd.concat([df[df['class'] == 1], df[df['class'] == 2]])
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])

# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
 ↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
train_set = paddedVec(w2v_google_model, train_set)
test_set = paddedVec(w2v_google_model, test_set)

## Substract 1 from labels, so the classes starts from 0
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
```

```
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)


## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

[20]:
```
model = SeqRNN(300,50,2)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0001)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):

        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
 ↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()
```

```
Epoch: 0 Timetaken: 35.71s train_loss: 0.4560 accuracy: 0.7240
Epoch: 1 Timetaken: 35.30s train_loss: 1.2912 accuracy: 0.7260
Epoch: 2 Timetaken: 37.21s train_loss: 0.3851 accuracy: 0.7475
Epoch: 3 Timetaken: 37.23s train_loss: 0.3212 accuracy: 0.7632
Epoch: 4 Timetaken: 36.77s train_loss: 0.3137 accuracy: 0.7430
Epoch: 5 Timetaken: 36.58s train_loss: 0.6706 accuracy: 0.7585
Epoch: 6 Timetaken: 36.34s train_loss: 0.3147 accuracy: 0.7370
Epoch: 7 Timetaken: 36.95s train_loss: 0.3153 accuracy: 0.7630
Epoch: 8 Timetaken: 36.17s train_loss: 1.2984 accuracy: 0.7582
Epoch: 9 Timetaken: 35.82s train_loss: 0.3610 accuracy: 0.7460
```

[21]:
```
torch.save(model,"binary_rnn_pre_trained.pkl")
```

[22]:
```
# Clear GPU Cache After Training
del train_set
del test_set
del train_labels
del test_labels
del training_set
del testing_set
del train_loader
del test_loader

del model
del loss_func
del optimizer
torch.cuda.empty_cache()
print(torch.cuda.memory_allocated())
print(torch.cuda.memory_reserved())
```

```
349696
2097152
```

0.8.3 Model #15. Self-trained W2V RNN-Simple-Ternary - PaddedVec

[23]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep all classes for Ternary Classification
df_sm = df.copy()
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])

# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
 ↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
```

```
train_set = paddedVec(w2v_review_model, train_set)
test_set = paddedVec(w2v_review_model, test_set)

## Substract 1 from labels, so the classes starts from 0
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)


## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

[24]:
```
# Ternary output_size = 3
model = SeqRNN(300,50,3)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0001)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):

        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()
```

```
Epoch: 0 Timetaken: 35.68s train_loss: 0.5820 accuracy: 0.5410
Epoch: 1 Timetaken: 36.97s train_loss: 0.5536 accuracy: 0.5905
Epoch: 2 Timetaken: 37.72s train_loss: 0.7429 accuracy: 0.6015
Epoch: 3 Timetaken: 37.33s train_loss: 0.5526 accuracy: 0.5900
Epoch: 4 Timetaken: 35.69s train_loss: 1.5254 accuracy: 0.6030
Epoch: 5 Timetaken: 36.47s train_loss: 1.5487 accuracy: 0.6135
Epoch: 6 Timetaken: 36.59s train_loss: 1.5013 accuracy: 0.6092
Epoch: 7 Timetaken: 37.87s train_loss: 0.6057 accuracy: 0.6172
Epoch: 8 Timetaken: 36.38s train_loss: 1.5396 accuracy: 0.6260
Epoch: 9 Timetaken: 36.91s train_loss: 1.4852 accuracy: 0.6192
```

[25]:
```
torch.save(model,"Ternary_rnn_self_trained.pkl")
```

0.8.4 Model #16. Pre-trained W2V RNN-Simple-Ternary - PaddedVec

[26]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep all classes for Ternary Classification
df_sm = df.copy()
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])

# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
 ↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
train_set = paddedVec(w2v_google_model, train_set)
test_set = paddedVec(w2v_google_model, test_set)

## Substract 1 from labels, so the classes starts from 0
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
```

```
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)


## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

[27]:
```
# Ternary output_size = 3
model = SeqRNN(300,50,3)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0001)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):

        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
 ↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()
```

```
Epoch: 0 Timetaken: 36.90s train_loss: 0.5729 accuracy: 0.5893
Epoch: 1 Timetaken: 37.07s train_loss: 1.5282 accuracy: 0.5625
Epoch: 2 Timetaken: 37.44s train_loss: 0.5607 accuracy: 0.5683
Epoch: 3 Timetaken: 38.09s train_loss: 1.2478 accuracy: 0.6042
Epoch: 4 Timetaken: 36.87s train_loss: 0.5780 accuracy: 0.6032
Epoch: 5 Timetaken: 40.22s train_loss: 1.5486 accuracy: 0.5995
Epoch: 6 Timetaken: 40.90s train_loss: 1.5482 accuracy: 0.5580
Epoch: 7 Timetaken: 38.51s train_loss: 1.2190 accuracy: 0.6035
Epoch: 8 Timetaken: 38.05s train_loss: 0.5546 accuracy: 0.6118
```

```
Epoch: 9 Timetaken: 37.31s train_loss: 1.5380 accuracy: 0.6148
```

[28]:
```
torch.save(model,"Ternary_rnn_pre_trained.pkl")
```

0.9 5b

[29]:
```
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

device = torch.device("cpu")

class GRU(nn.Module):

    def __init__(self,vocab_size,hidden_size,output_size):
        super(GRU,self).__init__()
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.gru = nn.GRU(self.vocab_size,self.hidden_size,batch_first=True)
        self.linear = nn.Linear(self.hidden_size,self.output_size)

    def forward(self,input):
        batch_size = len(input)
        h0 = torch.zeros(1,batch_size,self.hidden_size).to(device)
        output , hidden = self.gru(input,h0)
        output = output[ : ,-1, : ]
        output = self.linear(output)
        output = torch.nn.functional.softmax(output,dim=1)
        return output
```

0.9.1 Model #17. self-trained W2V RNN-GRU-Binary - PaddedVec

[31]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep only class 1 and 2 for Binary Classification
df_sm = pd.concat([df[df['class'] == 1], df[df['class'] == 2]])
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])
```

```
# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
train_set = paddedVec(w2v_review_model, train_set)
test_set = paddedVec(w2v_review_model, test_set)

## Substract 1 from labels, so the classes starts from 0
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)


## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

[32]:
```
# Ternary output_size = 2
model = SeqRNN(300,50,2)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.00005)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):
```

```
        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()
```

```
Epoch: 0 Timetaken: 37.67s train_loss: 0.7288 accuracy: 0.7522
Epoch: 1 Timetaken: 39.10s train_loss: 0.6166 accuracy: 0.7755
Epoch: 2 Timetaken: 37.98s train_loss: 0.3183 accuracy: 0.7792
Epoch: 3 Timetaken: 40.55s train_loss: 1.0035 accuracy: 0.7375
Epoch: 4 Timetaken: 38.89s train_loss: 0.3137 accuracy: 0.7742
Epoch: 5 Timetaken: 38.19s train_loss: 0.3176 accuracy: 0.7728
Epoch: 6 Timetaken: 39.31s train_loss: 1.3064 accuracy: 0.7505
Epoch: 7 Timetaken: 38.00s train_loss: 0.3147 accuracy: 0.7738
Epoch: 8 Timetaken: 38.17s train_loss: 0.3187 accuracy: 0.7827
Epoch: 9 Timetaken: 37.61s train_loss: 0.3133 accuracy: 0.7650
```

[33]:
```
torch.save(model,"Binary_gru_self_trained.pkl")
```

0.9.2 Model #18. pre-trained W2V RNN-GRU-Binary - PaddedVec

[34]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep only class 1 and 2 for Binary Classification
df_sm = pd.concat([df[df['class'] == 1], df[df['class'] == 2]])
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])

# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
```

```
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
train_set = paddedVec(w2v_google_model, train_set)
test_set = paddedVec(w2v_google_model, test_set)

## Substract 1 from labels, so the classes starts from 0
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)


## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

[35]:
```
# Ternary output_size = 2
model = SeqRNN(300,50,2)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.00005)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):

        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
```

```
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()
```

```
Epoch: 0 Timetaken: 38.48s train_loss: 1.1987 accuracy: 0.7435
Epoch: 1 Timetaken: 39.16s train_loss: 0.7993 accuracy: 0.7598
Epoch: 2 Timetaken: 38.48s train_loss: 0.3161 accuracy: 0.7685
Epoch: 3 Timetaken: 37.85s train_loss: 0.3135 accuracy: 0.7630
Epoch: 4 Timetaken: 37.58s train_loss: 0.3185 accuracy: 0.7798
Epoch: 5 Timetaken: 39.75s train_loss: 0.5101 accuracy: 0.7795
Epoch: 6 Timetaken: 39.49s train_loss: 0.3138 accuracy: 0.7840
Epoch: 7 Timetaken: 37.55s train_loss: 1.3017 accuracy: 0.7835
Epoch: 8 Timetaken: 37.73s train_loss: 0.3140 accuracy: 0.7770
Epoch: 9 Timetaken: 40.85s train_loss: 1.3131 accuracy: 0.7550
```

[36]:
```
torch.save(model,"Binary_gru_pre_trained.pkl")
```

0.9.3 Model #19. selftrained W2V RNN-GRU-Ternary - PaddedVec

[37]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep all classes for Ternary Classification
df_sm = df.copy()
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])

# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
train_set = paddedVec(w2v_review_model, train_set)
test_set = paddedVec(w2v_review_model, test_set)

## Substract 1 from labels, so the classes starts from 0
```

```
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)


## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

[38]:
```
# Ternary output_size = 3
model = SeqRNN(300,50,3)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0001)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):

        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()
```

```
Epoch: 0 Timetaken: 37.66s train_loss: 1.5264 accuracy: 0.5790
Epoch: 1 Timetaken: 40.00s train_loss: 1.5439 accuracy: 0.6132
Epoch: 2 Timetaken: 40.17s train_loss: 0.5606 accuracy: 0.6140
Epoch: 3 Timetaken: 38.97s train_loss: 1.4292 accuracy: 0.6010
```

```
Epoch: 4 Timetaken: 38.82s train_loss: 0.5570 accuracy: 0.6085
Epoch: 5 Timetaken: 38.93s train_loss: 0.5615 accuracy: 0.5877
Epoch: 6 Timetaken: 39.82s train_loss: 0.5525 accuracy: 0.6120
Epoch: 7 Timetaken: 38.25s train_loss: 0.7655 accuracy: 0.6080
Epoch: 8 Timetaken: 41.29s train_loss: 1.5390 accuracy: 0.6178
Epoch: 9 Timetaken: 39.71s train_loss: 1.5495 accuracy: 0.6185
```

[39]:
```
torch.save(model,"Ternary_gru_self_trained.pkl")
```

0.9.4 Model #20. Pre-trained W2V RNN-GRU-Ternary - PaddedVec

[40]:
```
## Cuda Device
device = None
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

## Keep all classes for Ternary Classification
df_sm = df.copy()
## Drop Null reviews
df_sm = df_sm.dropna(subset=['review'])

# Sample less data due to memory constraint
df_sm = df_sm.sample(20000)


# Split train and test data
reviews = df_sm["review"]
labels = df_sm["class"]
train_set, test_set, train_labels, test_labels = train_test_split(reviews,␣
 ↪labels, test_size = 0.2)

## Turn Train and Test set and labels to numpy array
train_set = paddedVec(w2v_google_model, train_set)
test_set = paddedVec(w2v_google_model, test_set)

## Substract 1 from labels, so the classes starts from 0
train_labels = train_labels.to_numpy() - 1
test_labels = test_labels.to_numpy() - 1

## Train & test tensors
train_set = torch.from_numpy(train_set).float().to(device)
test_set = torch.from_numpy(test_set).float().to(device)
train_labels = torch.from_numpy(train_labels).type(torch.LongTensor).to(device)
test_labels = torch.from_numpy(test_labels).type(torch.LongTensor).to(device)
```

```
## Prepare TensorDataset and DataLoader
training_set = TensorDataset(train_set, train_labels)
testing_set = TensorDataset(test_set, test_labels)

train_loader = DataLoader(training_set, batch_size=1, shuffle=True)
test_loader = DataLoader(testing_set, batch_size=1, shuffle=True)
```

[41]:
```
# Ternary output_size = 3
model = SeqRNN(300,50,3)
model.to(device)
epoches = 10
every_epoch = 1
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0001)

start_time = time.time()

for e in range(epoches):
    # Train all sentences from train set
    for i, (review, label) in enumerate(train_loader):

        optimizer.zero_grad()
        pred = model(review)
        loss = loss_func(pred,label)
        loss.backward()
        optimizer.step()

    if e % every_epoch == 0:
        accuracy = test_accuracy_loader(model, test_loader)
        timetaken = time.time() - start_time
        print("Epoch: {} Timetaken: {:.2f}s train_loss: {:.4f} accuracy: {:.
 ↪4f}".format(e, timetaken, loss, accuracy))
        start_time = time.time()
```

```
Epoch: 0 Timetaken: 38.34s train_loss: 0.5716 accuracy: 0.5857
Epoch: 1 Timetaken: 39.33s train_loss: 0.6329 accuracy: 0.6055
Epoch: 2 Timetaken: 37.41s train_loss: 0.5713 accuracy: 0.6060
Epoch: 3 Timetaken: 37.16s train_loss: 0.5569 accuracy: 0.6212
Epoch: 4 Timetaken: 37.89s train_loss: 0.5523 accuracy: 0.6092
Epoch: 5 Timetaken: 38.26s train_loss: 1.3709 accuracy: 0.6062
Epoch: 6 Timetaken: 38.46s train_loss: 1.5387 accuracy: 0.5675
Epoch: 7 Timetaken: 37.76s train_loss: 1.5017 accuracy: 0.6132
Epoch: 8 Timetaken: 39.33s train_loss: 1.1317 accuracy: 0.6122
Epoch: 9 Timetaken: 37.97s train_loss: 0.5584 accuracy: 0.6192
```

[42]:
```
torch.save(model,"Ternary_gru_pre_trained.pkl")
```

Conclusion: The accuracy for Binary and Ternary classification I got from GRU is very similar to the accuracy I got from Simple RNN models, with Binary Accuracy is higher than ternary accuracy.

The overall accuracy of RNN models is slightly lower than FNN models, I think the main reason is that the epochs and training set for RNN is smaller than what I have used in FNN, due to slow trainning speed and memory usage. For RNN, the entire dataset before train/test split is 20,000 reviews.

0.10 Reference Links

I've read and referenced some of the code from the following toturials and posts:
https://radimrehurek.com/gensim/models/word2vec.html

https://towardsdatascience.com/using-word2vec-to-analyze-news-headlines-and-predict-article-success-cdeda5f14751

https://stackoverflow.com/questions/10686924/numpy-array-to-scipy-sparse-matrix

https://stackoverflow.com/questions/29314033/drop-rows-containing-empty-cells-from-a-pandas-dataframe

https://blog.csdn.net/wuwususheng/article/details/109851179?utm_medium=distribute.pc_relevant.none-task-blog-2_{default}baidujs_title~default-4.no_search_link&spm=1001.2101.3001.4242

https://blog.csdn.net/sinat_33761963/article/details/104332831?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link

https://blog.csdn.net/Frierice/article/details/104286545?ops_request_misc=%257B%2522request%255Fid%2522
task-blog-2_{all}baidu_landing_v2~default-4-104286545.pc_search_result_hbase_insert&utm_term=pytorch+rnn-

https://medium.com/@martinpella/how-to-use-pre-trained-word-embeddings-in-pytorch-71ca59249f76

https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

https://www.kaggle.com/kanncaa1/recurrent-neural-network-with-pytorch