CSCI544 HW2

Yuhang Xiao - 6913860906 - yxiao776@usc.edu

Environment requirements are listed in requirements.txt

Report

Dataset Generation

Using the same data source as HW1 and the same cleaning and preprocessing procedures.

Word Embedding

Two examples of semantic similarity:

- China - Beijing + Tokyo = Japan
- big ~ huge

(a) Pretrained Word2Vec:

- China - Beijing + Tokyo = [('Japan', 0.8236700892448425)]
- big ~ huge: 0.7809856

(b) Trained Word2Vec:

- China - Beijing + Tokyo = [('Japan', 0.6667723655700684)]
- big ~ huge: 0.7743653

We can conclude that both models could encode the words close to their semantic similar words in the word embedding space. Our own trained Word2Vec could extract the same similar word as the pretrained model does, even though it's only trained on the review corpus. However, it seems that the pretrained model is still better because it achieves higher scores in similarity.

Simple Models

Binary Classfication Accuracy

Perceptron:

- Pretrained word2vec-google-news-300: 0.853225
- Trained Word2Vec: 0.8785
- TF-IDF: 0.930925

SVM:

- Pretrained word2vec-google-news-300: 0.8719
- Trained Word2Vec: 0.910625
- TF-IDF: 0.938175

We can conclude that using TF-IDF features gets the best performance, which may be because that we only keep the important words that matter for sentimental analysis and TF-IDF is better at capturing information of important words. Besides, a simple average pooling of each review for Word2Vec may harm its performance. Our own trained Word2Vec has higher accuracies than pretrained Word2Vec, which may be because our own Word2Vec is trained on the reviews directly that captures features better in the context of reviews.

Feedforward Neural Networks

Implementation: Flattened feature vectors go through two hidden layers with 50 and 10 nodes respectively, then a predicition head with 2 (for binary class) or 3 (for ternary class) is attached to predict the class label. Cross entropy loss is used and the FNN is optimized by AdamW optimizer with a cosine annealing learning rate scheduler.

(a) Average Pooling Feature

Binary classification accuracy:

- Pretrained Word2Vec: 0.90195
- Trained Word2Vec: 0.92155

Ternary classification accuracy:

- **Pretrained Word2Vec: 0.7614**
- **Trained Word2Vec: 0.79716**

(b) 10-word Sequence Feature

Binary classification accuracy:

- **Pretrained Word2Vec: 0.85345**
- **Trained Word2Vec: 0.8655**

Ternary classification accuracy:

- **Pretrained Word2Vec: 0.70042**
- **Trained Word2Vec: 0.7196**

By using FFN models, word2vec features can get better results compared to using them on simple models, such as Perceptron and SVM. When using the same average pooling strategy, word2vec features on FFN models can get very close preformances compared to TF-IDF features on simple models. When using the first 10 words concatenation strategy, the performances are worse, which may be because the features don't capture the information of the whole review. Besides, our own trained word2vec gets better results than pretrained word2vec as in simple models.

Convolutional Neural Networks

Implementation: Feature vectors with a sequence length of 50 first go through two CNN layers with 50 and 10 output channels respectively, which are both using kernels with kernel size = 3, padding = 1, stride = 1 so that the sequence length is preserved. Then a maxpooling layer is applied to squeeze the sequence length. Finally, a predicition head with 2 (for binary class) or 3 (for ternary class) is attached to predict the class label. Cross entropy loss is used and the CNN is optimized by AdamW optimizer with a cosine annealing learning rate scheduler.

Binary classification accuracy:

- **Pretrained Word2Vec: 0.919525**
- **Trained Word2Vec: 0.9204**

Ternary classification accuracy:

- **Pretrained Word2Vec: 0.77694**
- **Trained Word2Vec: 0.78948**

In [284]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
import re
from bs4 import BeautifulSoup
import contractions
import warnings
```

```
[nltk_data] Downloading package wordnet to /lab/mydcxiao/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /lab/mydcxiao/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /lab/mydcxiao/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Read Data

In [2]:

```
dtype={7: object}
url = 'https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazo
n_reviews_us_Office_Products_v1_00.tsv.gz'
data = pd.read_csv(url, sep='\t', on_bad_lines='skip', dtype=dtype)
# path = 'amazon_reviews_us_Office_Products_v1_00.tsv.gz'
# data = pd.read_csv(path, sep='\t', on_bad_lines='skip', dtype=dtype)
# unzipped_path = 'amazon_reviews_us_Office_Products_v1_00.tsv'
# data = pd.read_csv(unzipped_path, sep='\t', on_bad_lines='skip', dtype=dtype)
print("Rows: ", data.shape[0])
```

```
data.head()
```

```
Rows:  2640254
```

Out[2]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_vote
0	US	43081963	R18RVCKGH1SSI9	B001BM2MAC	307809868	Scotch Cushion Wrap 7961, 12 Inches x 100 Feet	Office Products	5	0.
1	US	10951564	R3L4L6LW1PUOFY	B00DZYEXPQ	75004341	Dust-Off Compressed Gas Duster, Pack of 4	Office Products	5	0.
2	US	21143145	R2J8AWXWTDX2TF	B00RTMUHDW	529689027	Amram Tagger Standard Tag Attaching Tagging Gu...	Office Products	5	0.
3	US	52782374	R1PR37BR7G3M6A	B00D7H8XB6	868449945	AmazonBasics 12-Sheet High-Security Micro-Cut ...	Office Products	1	2.
4	US	24045652	R3BDDDZMZBZDPU	B001XCWP34	33521401	Derwent Colored Pencils, Inktense Ink Pencils,...	Office Products	4	0.

Keep Reviews and Ratings

In [3]:

```
data = data[['star_rating', 'review_headline', 'review_body']]
data.dropna(inplace=True)
print("Rows: ", data.shape[0])
print("Three sample reviews:")
data.sample(3, random_state=1)
```

```
Rows:  2640037
Three sample reviews:
```

Out[3]:

	star_rating	review_headline	review_body
246582	5	Fast Shipping!	Fast Shipping. Works Perfectly!
2639050	5	it works!	5 stars based on transmission clarity. compar...
697475	5	Five Stars	Love it, looks great on my wall!

Data Statistics

In [4]:

```
data['star_rating'] = pd.to_numeric(data['star_rating'], errors='coerce')
print("All reviews: ", data.shape[0])
print(f"1 rating reviews: {data[data['star_rating'] == 1].shape[0]}, {100 * data[data['star_rating'] == 1]
.shape[0]/data.shape[0]}%")
print(f"2 rating reviews: {data[data['star_rating'] == 2].shape[0]}, {100 * data[data['star_rating'] == 2]
.shape[0]/data.shape[0]}%")
print(f"3 rating reviews: {data[data['star_rating'] == 3].shape[0]}, {100 * data[data['star_rating'] == 3]
.shape[0]/data.shape[0]}%")
print(f"4 rating reviews: {data[data['star_rating'] == 4].shape[0]}, {100 * data[data['star_rating'] == 4]
.shape[0]/data.shape[0]}%")
print(f"5 rating reviews: {data[data['star_rating'] == 5].shape[0]}, {100 * data[data['star_rating'] == 5]
.shape[0]/data.shape[0]}%")
```

```
All reviews:  2640037
1 rating reviews: 306962, 11.627185528081615%
2 rating reviews: 138380, 5.241593204943719%
3 rating reviews: 193674, 7.336033548014668%
4 rating reviews: 418339, 15.845952159003833%
```

```
5 rating reviews: 1582682, 59.949235559956165%
```

We form three classes and select 50000 reviews randomly from each star ratings.

In [5]:

```
pos_reviews = data[data['star_rating'] > 3]
neg_reviews = data[data['star_rating'] <= 2]
neu_reviews = data[data['star_rating'] == 3]
print("All reviews: ", data.shape[0])
print("Positive reviews: ", pos_reviews.shape[0])
print("Negative reviews: ", neg_reviews.shape[0])
print("Neutral reviews: ", neu_reviews.shape[0])
print("All reviews == Positive + Negative + Neutral: ", data.shape[0] == pos_reviews.shape[0] + neg_revie
ws.shape[0] + neu_reviews.shape[0])
```

```
All reviews:  2640037
Positive reviews:  2001021
Negative reviews:  445342
Neutral reviews:  193674
All reviews == Positive + Negative + Neutral:  True
```

In [6]:

```
pos_samples = pos_reviews.sample(n=100000, random_state=0)
neg_samples = neg_reviews.sample(n=100000, random_state=0)
neu_samples = neu_reviews.sample(n=50000, random_state=0)
pos_samples['star_rating'] = 1
neg_samples['star_rating'] = 2
neu_samples['star_rating'] = 3
pos_samples.rename(columns={'star_rating': 'label'}, inplace=True)
neg_samples.rename(columns={'star_rating': 'label'}, inplace=True)
neu_samples.rename(columns={'star_rating': 'label'}, inplace=True)
dataset = pd.concat([pos_samples, neg_samples, neu_samples])
dataset['review'] = dataset[['review_headline', 'review_body']].agg(' '.join, axis=1)
dataset.drop(columns=['review_headline', 'review_body'], inplace=True)
print("Rows: ", dataset.shape[0])
# print("Three sample reviews:")
# dataset.sample(3, random_state=1)
```

```
Rows:  250000
```

Data Cleaning

In [7]:

```
print("Average length of reviews before cleaning: ", dataset['review'].str.len().mean())
print("Three sample reviews:")
dataset.sample(3,random_state=1)
```

```
Average length of reviews before cleaning:  353.398672
Three sample reviews:
```

Out[7]:

	label	review
508714	3	Good Phone but Major Design Flaw The Phones th...
713271	1	Five Stars Great
3017	3	Three Stars It's a little less sturdy than the...

In [8]:

```
from bs4 import MarkupResemblesLocatorWarning
warnings.filterwarnings("ignore", category=MarkupResemblesLocatorWarning)
dataset['review'] = dataset['review'].apply(str.lower)
dataset['review'] = dataset['review'].apply(lambda x: re.sub(r'https?://\S+|www\.\S+', '', x))
dataset['review'] = dataset['review'].apply(lambda x: BeautifulSoup(x, "html.parser").text)
dataset['review'] = dataset['review'].apply(lambda x: re.sub(r'[^a-zA-Z]+', ' ', x))
dataset['review'] = dataset['review'].apply(lambda x: re.sub(r'\s+', ' ', x).strip())
dataset['review'] = dataset['review'].apply(lambda x: ' '.join([contractions.fix(word) for word in x.spli
t()]))

print("Average length of reviews after cleaning: ", dataset['review'].str.len().mean())
print("Three sample reviews:")
dataset.sample(3,random_state=1)
```

```
Average length of reviews after cleaning:  335.85226
Three sample reviews:
```

Out[8]:

	label	review
508714	3	good phone but major design flaw the phones th...
713271	1	five stars great
3017	3	three stars it s a little less sturdy than the...

Pre-processing

In [9]:

```
print("Average length of reviews before preprocessing: ", dataset['review'].str.len().mean())
print("Three sample reviews:")
dataset.sample(3,random_state=1)
```

```
Average length of reviews before preprocessing:  335.85226
Three sample reviews:
```

Out[9]:

	label	review
508714	3	good phone but major design flaw the phones th...
713271	1	five stars great
3017	3	three stars it s a little less sturdy than the...

remove the stop words

In [10]:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stopwords = set(stopwords.words('english'))
dataset['review'] = dataset['review'].apply(lambda x: ' '.join([word for word in word_tokenize(x) if word
not in stopwords]))
print("Three sample reviews:")
dataset.sample(3,random_state=1)
```

```
Three sample reviews:
```

Out[10]:

	label	review
508714	3	good phone major design flaw phones work good ...
713271	1	five stars great
3017	3	three stars little less sturdy previous versio...

perform lemmatization

In [11]:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
dataset['review'] = dataset['review'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word) for word in wo
rd_tokenize(x)]))
print("Average length of reviews after preprocessing: ", dataset['review'].str.len().mean())
print("Three sample reviews:")
dataset.sample(3,random_state=1)
```

```
Average length of reviews after preprocessing:  209.386508
Three sample reviews:
```

Out[11]:

	label	review
508714	3	good phone major design flaw phone work good f...

	label	review
713271	1	five star great
3017	3	three star little le sturdy previous version b...

Word Embedding

(a) Pretrained Word2Vec

In [12]:

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

Two examples of semantic similarities \ (1) China - Beijing + Tokyo = Japan \ (2) big ~ huge

In [47]:

```
# example 1
print("China - Beijing + Tokyo =", wv.most_similar(positive=['China', 'Tokyo'], negative=['Beijing'], top
n=1))
# example 2
print("big ~ huge:", wv.similarity('big', 'huge'))
```

```
China - Beijing + Tokyo = [('Japan', 0.8236700892448425)]
big ~ huge: 0.7809856
```

(b) Trained Word2Vec

In [20]:

```
import gensim.models
corpus = data[['review_headline', 'review_body']]
corpus['review'] = corpus[['review_headline', 'review_body']].agg(' '.join, axis=1)
corpus.drop(columns=['review_headline', 'review_body'], inplace=True)
corpus['review'] = corpus['review'].apply(str)
model = gensim.models.Word2Vec(corpus['review'].apply(word_tokenize), vector_size=300, window=11, min_cou
nt=10)
```

Comparison of the two examples with pretrained model

In [48]:

```
# example 1
print("China - Beijing + Tokyo =", model.wv.most_similar(positive=['China', 'Tokyo'], negative=['Beijing'
], topn=1))
# example 2
print("big ~ huge:", model.wv.similarity('big', 'huge'))
```

```
China - Beijing + Tokyo = [('Japan', 0.6667723655700684)]
big ~ huge: 0.7743653
```

We can conclude that both models could encode the words close to their semantic similar words in the word embedding space. Our own trained Word2Vec could extract the same similar word as the pretrained model does, even though it's only trained on the review corpus. However, it seems that the pretrained model is still better because it achieves higher scores in similarity.

Simple Models

Only use class 1 and class 2 for binary classification.

TF-IDF Feature Extraction

In [71]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1, 3))
vectors = vectorizer.fit_transform(dataset['review'][dataset['label'] != 3])
print(vectors.shape)
```

```
(200000, 6740105)
```

Train-Test Split

In [72]:

```
from sklearn.model_selection import train_test_split
labels = dataset['label'][dataset['label'] != 3]
reviews = dataset['review'][dataset['label'] != 3]
corpus_train, corpus_test, x_train, x_test, y_train, y_test = train_test_split(reviews, vectors, labels,
test_size=0.2, random_state=42, stratify=labels)
print("Training set size: ", corpus_train.shape, x_train.shape, y_train.shape)
print("Test set size: ", corpus_test.shape, x_test.shape, y_test.shape)
```

```
Training set size:  (160000,) (160000, 6740105) (160000,)
Test set size:  (40000,) (40000, 6740105) (40000,)
```

Average the Corpus Using Word2Vec

In [111]:

```
def parse_then_average(wv, sentence):
    words = word_tokenize(sentence)
    words = [word for word in words if word in wv.key_to_index]
    if len(words) == 0:
        return np.zeros(wv.vector_size)
    return np.mean(wv[words], axis=0)

pretrained_word_embeddings = np.vstack(corpus_train.apply(lambda x: parse_then_average(wv, x)))
trained_word_embeddings = np.vstack(corpus_train.apply(lambda x: parse_then_average(model.wv, x)))
pretrained_word_embeddings_test = np.vstack(corpus_test.apply(lambda x: parse_then_average(wv, x)))
trained_word_embeddings_test = np.vstack(corpus_test.apply(lambda x: parse_then_average(model.wv, x)))
print(pretrained_word_embeddings.shape, trained_word_embeddings.shape)
print(pretrained_word_embeddings_test.shape, trained_word_embeddings_test.shape)
```

```
(160000, 300) (160000, 300)
(40000, 300) (40000, 300)
```

Perceptron

In [275]:

```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import Perceptron
perceptron1 = Perceptron(tol=1e-2)
perceptron2 = Perceptron()
perceptron3 = Perceptron()
perceptron1.fit(pretrained_word_embeddings, y_train)
perceptron2.fit(trained_word_embeddings, y_train)
perceptron3.fit(x_train, y_train)
pred_test1 = perceptron1.predict(pretrained_word_embeddings_test)
pred_test2 = perceptron2.predict(trained_word_embeddings_test)
pred_test3 = perceptron3.predict(x_test)
print("Pretrained word2vec-google-news-300:", accuracy_score(y_test, pred_test1))
print("Trained Word2Vec:", accuracy_score(y_test, pred_test2))
print("TF-IDF:", accuracy_score(y_test, pred_test3))
```

```
Pretrained word2vec-google-news-300: 0.853225
Trained Word2Vec: 0.8785
TF-IDF: 0.930925
```

SVM

In [113]:

```
from sklearn.svm import LinearSVC
svm1 = LinearSVC(dual='auto', random_state=0)
svm2 = LinearSVC(dual='auto', random_state=0)
svm3 = LinearSVC(dual='auto', random_state=0)
svm1.fit(pretrained_word_embeddings, y_train)
svm2.fit(trained_word_embeddings, y_train)
svm3.fit(x_train, y_train)
pred_test1 = svm1.predict(pretrained_word_embeddings_test)
pred_test2 = svm2.predict(trained_word_embeddings_test)
pred_test3 = svm3.predict(x_test)
print("Pretrained word2vec-google-news-300:", accuracy_score(y_test, pred_test1))
```

```
print("Trained Word2Vec:", accuracy_score(y_test, pred_test2))
print("TF-IDF:", accuracy_score(y_test, pred_test3))
```

```
Pretrained word2vec-google-news-300: 0.8719
Trained Word2Vec: 0.910625
TF-IDF: 0.938175
```

We can conclude that using TF-IDF features gets the best performance, which may be because that we only keep the important words of the reviews and TF-IDF is better at capturing information of important words. Besides, a simple average pooling of each review for Word2Vec may harm its performance. Our own trained Word2Vec has higher accuracies than pretrained Word2Vec, which may be because our own Word2Vec is trained on the reviews directly that captures features better in the context of reviews.

Feedforward Neural Networks

Train-Test Split

In [161]:

```
labels = dataset['label']
reviews = dataset['review']
reviews_train, reviews_test, labels_train, labels_test = train_test_split(reviews, labels, test_size=0.2,
random_state=42, stratify=labels)
print("Training set size: ", reviews_train.shape, labels_train.shape)
print("Test set size: ", reviews_test.shape, labels_test.shape)
```

```
Training set size:  (200000,) (200000,)
Test set size:  (50000,) (50000,)
```

Dataset Class

In [223]:

```
import torch
from torch.utils.data import Dataset, DataLoader

class ReviewsDataset(Dataset):
    def __init__(self, reviews, labels, wv, max_seq_len=10, average_pooling=True):
        self.reviews = reviews
        self.labels = labels
        self.wv = wv
        self.average_pooling = average_pooling
        self.max_seq_len = max_seq_len
    def __len__(self):
        return self.reviews.shape[0]
    def __getitem__(self, idx):
        if self.average_pooling:
            sent_embedding = torch.from_numpy(parse_then_average(self.wv, self.reviews.iloc[idx])).unsque
eze(0) # 1 x 300
        else:
            sentence = word_tokenize(self.reviews.iloc[idx])
            word_embedding_list = []
            i, j = 0, 0
            while i < self.max_seq_len:
                if j >= len(sentence):
                    word_embedding_list.append(torch.zeros(self.wv.vector_size))
                    i += 1
                else:
                    if sentence[j] in self.wv.key_to_index:
                        word_embedding_list.append(torch.from_numpy(self.wv[sentence[j]].copy()))
                        i += 1
                        j += 1
                    else:
                        j += 1
            sent_embedding = torch.stack(word_embedding_list) # 10 x 300
        return sent_embedding.float(), self.labels.iloc[idx] - 1
```

Engine

In [180]:

```
from tqdm import tqdm

def train_model(model, dataloader, criterion, optimizer, scheduler, num_epochs=10):
    device = next(model.parameters()).device
```

```python
    for epoch in tqdm(range(num_epochs), desc="Training"):
        model.train()
        avg_loss = 0.0
        for batch in dataloader:
            inputs, labels = batch
            inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            scheduler.step()
            avg_loss += loss.item()
        if epoch % 10 == 0:
            print(f"Epoch {epoch+1}, loss: {avg_loss / len(dataloader)}")

def test_model(model, dataloader):
    model.eval()
    correct = 0
    total = 0
    device = next(model.parameters()).device
    with torch.no_grad():
        for batch in tqdm(dataloader, desc="Testing"):
            inputs, labels = batch
            inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True)
            outputs = model(inputs)
            _, pred = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (pred == labels).sum().item()
    return correct / total
```

(a) Use Average Pooling Feature

Binary Classification

Pretrained Word2Vec

In [190]:

```python
num_epochs = 30
batch_size = 500

BinaryPretrainedAvgPoolingDataset = ReviewsDataset(reviews_train[labels_train != 3], labels_train[labels_
train != 3], wv)
BinaryPretrainedAvgPoolingDataloader = DataLoader(BinaryPretrainedAvgPoolingDataset, batch_size=batch_size
, shuffle=True, num_workers=24)

input_dim = BinaryPretrainedAvgPoolingDataset[0][0].shape[1]

binary_mlp_1 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 2),
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
binary_mlp_1 = binary_mlp_1.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(binary_mlp_1.parameters(), lr=0.01)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(BinaryPretraine
dAvgPoolingDataloader) // batch_size, eta_min=1e-6)

train_model(binary_mlp_1, BinaryPretrainedAvgPoolingDataloader, criterion, optimizer, lr_scheduler, num_e
pochs=num_epochs)
```

```
Training:   0%|          | 0/30 [00:00<?, ?it/s]Training:   3%|▎         | 1/30 [00:13<06:19, 13.10s/it]
```

```
Epoch 1, loss: 0.3314122884068638
```

```
Training:  37%|███▋      | 11/30 [02:11<03:41, 11.64s/it]
```

```
Epoch 11, loss: 0.22371997023001314
```

```
Training:  70%|███████   | 21/30 [04:07<01:44, 11.60s/it]
```

```
Epoch 21, loss: 0.2046239526476711
```

```
Training: 100%|██████████| 30/30 [05:54<00:00, 11.80s/it]
```

In [191]:

```
BinaryPretrainedAvgPoolingDataloader_test = DataLoader(ReviewsDataset(reviews_test[labels_test != 3], labe
ls_test[labels_test != 3], wv),
                                                  batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(binary_mlp_1, BinaryPretrainedAvgPoolingDataloader_test))
```

```
Testing:   0%|          | 0/80 [00:00<?, ?it/s]Testing: 100%|██████████| 80/80 [00:08<00:00,  9.25it/s]
```

```
Classification Accuracy: 0.90195
```

Trained Word2Vec

In [196]:

```
num_epochs = 30
batch_size = 500

BinaryTrainedAvgPoolingDataset = ReviewsDataset(reviews_train[labels_train != 3], labels_train[labels_tra
in != 3], model.wv)
BinaryTrainedAvgPoolingDataloader = DataLoader(BinaryTrainedAvgPoolingDataset, batch_size=batch_size, shu
ffle=True, num_workers=24)

input_dim = BinaryTrainedAvgPoolingDataset[0][0].shape[1]

binary_mlp_2 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 2),
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
binary_mlp_2 = binary_mlp_2.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(binary_mlp_2.parameters(), lr=0.01)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(BinaryTrainedAv
gPoolingDataloader) // batch_size, eta_min=1e-6)

train_model(binary_mlp_2, BinaryTrainedAvgPoolingDataloader, criterion, optimizer, lr_scheduler, num_epoc
hs=num_epochs)
```

```
Training:   0%|          | 0/30 [00:00<?, ?it/s]Training:   3%|▎         | 1/30 [00:12<06:09, 12.76s/it]
```

```
Epoch 1, loss: 0.225943755870685
```

```
Training:  37%|███▋      | 11/30 [02:13<03:55, 12.38s/it]
```

```
Epoch 11, loss: 0.1656110317679122
```

```
Training:  70%|███████   | 21/30 [04:11<01:43, 11.53s/it]
```

```
Epoch 21, loss: 0.1485034336335957
```

```
Training: 100%|██████████| 30/30 [05:58<00:00, 11.95s/it]
```

In [197]:

```
BinaryTrainedAvgPoolingDataloader_test = DataLoader(ReviewsDataset(reviews_test[labels_test != 3], labels
_test[labels_test != 3], model.wv),
                                                  batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(binary_mlp_2, BinaryTrainedAvgPoolingDataloader_test))
```

```
Testing:   0%|          | 0/80 [00:00<?, ?it/s]Testing: 100%|██████████| 80/80 [00:09<00:00,  8.49it/s]
```

```
Classification Accuracy: 0.92155
```

Ternary Classification

Pretrained Word2Vec

In [216]:

```
num_epochs = 30
batch_size = 500

TernaryPretrainedAvgPoolingDataset = ReviewsDataset(reviews_train, labels_train, wv)
```

```python
TernaryPretrainedAvgPoolingDataloader = DataLoader(TernaryPretrainedAvgPoolingDataset, batch_size=batch_si
ze, shuffle=True, num_workers=24)

input_dim = TernaryPretrainedAvgPoolingDataset[0][0].shape[1]

ternary_mlp_1 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 3),
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ternary_mlp_1 = ternary_mlp_1.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(ternary_mlp_1.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(TernaryPretrain
edAvgPoolingDataloader) // batch_size, eta_min=1e-8)

train_model(ternary_mlp_1, TernaryPretrainedAvgPoolingDataloader, criterion, optimizer, lr_scheduler, num
_epochs=num_epochs)
```

```
Training:   0%|          | 0/30 [00:00<?, ?it/s]Training:   3%|▎         | 1/30 [00:13<06:32, 13.55s/it]
```

```
Epoch 1, loss: 0.8390966232120991
```

```
Training:  37%|███▋      | 11/30 [02:38<04:37, 14.61s/it]
```

```
Epoch 11, loss: 0.581205048263073
```

```
Training:  70%|███████   | 21/30 [05:02<02:09, 14.39s/it]
```

```
Epoch 21, loss: 0.5530290900915861
```

```
Training: 100%|██████████| 30/30 [07:09<00:00, 14.33s/it]
```

In [217]:

```python
TernaryPretrainedAvgPoolingDataloader_test = DataLoader(ReviewsDataset(reviews_test, labels_test, wv),
                                                  batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(ternary_mlp_1, TernaryPretrainedAvgPoolingDataloader_test))
```

```
Testing: 100%|██████████| 100/100 [00:11<00:00,  8.64it/s]
```

```
Classification Accuracy: 0.7614
```

Trained Word2Vec

In [218]:

```python
num_epochs = 30
batch_size = 500

TernaryTrainedAvgPoolingDataset = ReviewsDataset(reviews_train, labels_train, model.wv)
TernaryTrainedAvgPoolingDataloader = DataLoader(TernaryTrainedAvgPoolingDataset, batch_size=batch_size, sh
uffle=True, num_workers=24)

input_dim = TernaryTrainedAvgPoolingDataset[0][0].shape[1]

ternary_mlp_2 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 3),
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ternary_mlp_2 = ternary_mlp_2.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(ternary_mlp_2.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(TernaryTrainedA
vgPoolingDataloader) // batch_size, eta_min=1e-8)

train_model(ternary_mlp_2, TernaryTrainedAvgPoolingDataloader, criterion, optimizer, lr_scheduler, num_ep
ochs=num_epochs)
```

```
Training:   3%|▎         | 1/30 [00:15<07:17, 15.09s/it]
```

```
Epoch 1, loss: 0.6173438151180745
```

```
Training:  37%|███▋      | 11/30 [02:43<04:43, 14.90s/it]
Epoch 11, loss: 0.4713097733259201
Training:  70%|███████   | 21/30 [05:12<02:13, 14.87s/it]
Epoch 21, loss: 0.4543842290341854
Training: 100%|██████████| 30/30 [07:19<00:00, 14.64s/it]
```

In [219]:

```python
TernaryTrainedAvgPoolingDataloader_test = DataLoader(ReviewsDataset(reviews_test, labels_test, model.wv),
                                                     batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(ternary_mlp_2, TernaryTrainedAvgPoolingDataloader_test))
```

```
Testing: 100%|██████████| 100/100 [00:09<00:00, 10.14it/s]
Classification Accuracy: 0.79716
```

(b) Use Concatenated Features

Binary Classification

Pretrained Word2Vec

In [228]:

```python
num_epochs = 30
batch_size = 500

BinaryPretrainedConcatDataset = ReviewsDataset(reviews_train[labels_train != 3], labels_train[labels_train
!= 3], wv, max_seq_len=10, average_pooling=False)
BinaryPretrainedConcatDataloader = DataLoader(BinaryPretrainedConcatDataset, batch_size=batch_size, shuff
le=True, num_workers=24)

input_dim = BinaryPretrainedConcatDataset[0][0].shape[1] * BinaryPretrainedConcatDataset[0][0].shape[0]

binary_mlp_3 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 2),
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
binary_mlp_3 = binary_mlp_3.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(binary_mlp_3.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(BinaryPretraine
dConcatDataloader) // batch_size, eta_min=1e-6)

train_model(binary_mlp_3, BinaryPretrainedConcatDataloader, criterion, optimizer, lr_scheduler, num_epoch
s=num_epochs)
```

```
Training:   0%|          | 0/30 [00:00<?, ?it/s]Training:   3%|▎         | 1/30 [00:13<06:28, 13.40s/it]
Epoch 1, loss: 0.3983966983854771
Training:  37%|███▋      | 11/30 [02:20<04:01, 12.73s/it]
Epoch 11, loss: 0.23861584621481596
Training:  70%|███████   | 21/30 [04:27<01:53, 12.60s/it]
Epoch 21, loss: 0.1779147365130484
Training: 100%|██████████| 30/30 [06:22<00:00, 12.76s/it]
```

In [229]:

```python
BinaryPretrainedConcatDataloader_test = DataLoader(ReviewsDataset(reviews_test[labels_test != 3], labels_
test[labels_test != 3], wv, max_seq_len=10, average_pooling=False),
                                                   batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(binary_mlp_3, BinaryPretrainedConcatDataloader_test))
```

```
Testing:   0%|          | 0/80 [00:00<?, ?it/s]Testing: 100%|██████████| 80/80 [00:10<00:00,  7.62it/s]
```

```
Classification Accuracy: 0.85345
```

Trained Word2Vec

In [230]:

```python
num_epochs = 30
batch_size = 500

BinaryTrainedConcatDataset = ReviewsDataset(reviews_train[labels_train != 3], labels_train[labels_train != 3], model.wv, max_seq_len=10, average_pooling=False)
BinaryTrainedConcatDataloader = DataLoader(BinaryTrainedConcatDataset, batch_size=batch_size, shuffle=True, num_workers=24)

input_dim = BinaryTrainedConcatDataset[0][0].shape[1] * BinaryTrainedConcatDataset[0][0].shape[0]

binary_mlp_4 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 2),
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
binary_mlp_4 = binary_mlp_4.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(binary_mlp_4.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(BinaryTrainedConcatDataloader) // batch_size, eta_min=1e-6)

train_model(binary_mlp_4, BinaryTrainedConcatDataloader, criterion, optimizer, lr_scheduler, num_epochs=num_epochs)
```

```
Training:   3%|▎         | 1/30 [00:12<06:00, 12.44s/it]
Epoch 1, loss: 0.29990119780413804
Training:  37%|███▋      | 11/30 [02:21<04:07, 13.01s/it]
Epoch 11, loss: 0.08220909271622076
Training:  70%|███████   | 21/30 [04:28<01:54, 12.67s/it]
Epoch 21, loss: 0.024188642197987064
Training: 100%|██████████| 30/30 [06:24<00:00, 12.81s/it]
```

In [231]:

```python
BinaryTrainedConcatDataloader_test = DataLoader(ReviewsDataset(reviews_test[labels_test != 3], labels_test[labels_test != 3], model.wv, max_seq_len=10, average_pooling=False),
                                                 batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(binary_mlp_4, BinaryTrainedConcatDataloader_test))
```

```
Testing: 100%|██████████| 80/80 [00:09<00:00,  8.34it/s]
Classification Accuracy: 0.8655
```

Ternary Classification

Pretrained Word2Vec

In [232]:

```python
num_epochs = 30
batch_size = 500

TernaryPretrainedConcatDataset = ReviewsDataset(reviews_train, labels_train, wv, max_seq_len=10, average_pooling=False)
TernaryPretrainedConcatDataloader = DataLoader(TernaryPretrainedConcatDataset, batch_size=batch_size, shuffle=True, num_workers=24)

input_dim = TernaryPretrainedConcatDataset[0][0].shape[1] * TernaryPretrainedConcatDataset[0][0].shape[0]

ternary_mlp_3 = nn.Sequential(
```

```
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 3),
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ternary_mlp_3 = ternary_mlp_3.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(ternary_mlp_3.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(TernaryPretrain
edConcatDataloader) // batch_size, eta_min=1e-8)

train_model(ternary_mlp_3, TernaryPretrainedConcatDataloader, criterion, optimizer, lr_scheduler, num_epo
chs=num_epochs)
```

```
Training:   3%|▎         | 1/30 [00:13<06:37, 13.70s/it]
Epoch 1, loss: 0.7570322492718696
Training:  37%|███▋      | 11/30 [02:31<04:18, 13.61s/it]
Epoch 11, loss: 0.447182969301939
Training:  70%|███████   | 21/30 [04:48<02:02, 13.62s/it]
Epoch 21, loss: 0.31888479202985764
Training: 100%|██████████| 30/30 [06:52<00:00, 13.77s/it]
```

In [233]:

```
TernaryPretrainedConcatDataloader_test = DataLoader(ReviewsDataset(reviews_test, labels_test, wv, max_seq
_len=10, average_pooling=False),
                                                     batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(ternary_mlp_3, TernaryPretrainedConcatDataloader_test))
```

```
Testing: 100%|██████████| 100/100 [00:09<00:00, 10.06it/s]
Classification Accuracy: 0.70042
```

Trained Word2Vec

In [234]:

```
num_epochs = 30
batch_size = 500

TernaryTrainedConcatDataset = ReviewsDataset(reviews_train, labels_train, model.wv, max_seq_len=10, avera
ge_pooling=False)
TernaryTrainedConcatDataloader = DataLoader(TernaryTrainedConcatDataset, batch_size=batch_size, shuffle=T
rue, num_workers=24)

input_dim = TernaryTrainedConcatDataset[0][0].shape[1] * TernaryTrainedConcatDataset[0][0].shape[0]

ternary_mlp_4 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(input_dim, 50),
    nn.ReLU(),
    nn.Linear(50, 10),
    nn.ReLU(),
    nn.Linear(10, 3),
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ternary_mlp_4 = ternary_mlp_4.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(ternary_mlp_4.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(TernaryTrainedC
oncatDataloader) // batch_size, eta_min=1e-8)

train_model(ternary_mlp_4, TernaryTrainedConcatDataloader, criterion, optimizer, lr_scheduler, num_epochs
=num_epochs)
```

```
Training:   3%|▎         | 1/30 [00:14<07:00, 14.50s/it]
Epoch 1, loss: 0.6140345711261034
Training:  37%|███▋      | 11/30 [02:31<04:24, 13.93s/it]
```

```
Epoch 11, loss: 0.3593694458156824
```

```
Training:  70%|███████   | 21/30 [04:49<02:05, 13.93s/it]
```

```
Epoch 21, loss: 0.23860658537596463
```

```
Training: 100%|██████████| 30/30 [06:53<00:00, 13.80s/it]
```

In [235]:

```python
TernaryTrainedConcatDataloader_test = DataLoader(ReviewsDataset(reviews_test, labels_test, model.wv, max_seq_len=10, average_pooling=False),
                                                    batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(ternary_mlp_4, TernaryTrainedConcatDataloader_test))
```

```
Testing: 100%|██████████| 100/100 [00:11<00:00,  8.82it/s]
```

```
Classification Accuracy: 0.7196
```

By using FFN models, word2vec features can get better results compared to using them on simple models, such as Perceptron and SVM. When using the same average pooling strategy, word2vec features on FFN models can get very close preformances compared to TF-IDF features on simple models. When using the first 10 words concatenation strategy, the performances are worse, which may be because the features don't capture the information of the whole review. Besides, our own trained word2vec gets better results than pretrained word2vec as in simple models.

Convolution Nerual Networks

Prepare CNN Model Class

In [236]:

```python
class CNN(nn.Module):
    def __init__(self, input_dim, output_dim, activation=nn.GELU):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(input_dim, 50, kernel_size=3, padding=1),
            activation(),
            nn.Conv1d(50, 10, kernel_size=3, padding=1),
            activation(),
            nn.AdaptiveMaxPool1d(1),
        )
        self.head = nn.Linear(10, output_dim)
    def forward(self, x):
        x = x.permute(0, 2, 1)
        x = self.cnn(x)
        x = x.squeeze(-1)
        return self.head(x)
```

Binary Classification

Pretrained Word2Vec

In [237]:

```python
num_epochs = 30
batch_size = 500

BinaryPretrainedCNNDataset = ReviewsDataset(reviews_train[labels_train != 3], labels_train[labels_train != 3], wv, max_seq_len=50, average_pooling=False)
BinaryPretrainedCNNDataloader = DataLoader(BinaryPretrainedCNNDataset, batch_size=batch_size, shuffle=True, num_workers=24)

input_dim = BinaryPretrainedCNNDataset[0][0].shape[1]

binary_cnn_1 = CNN(input_dim, 2)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
binary_cnn_1 = binary_cnn_1.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(binary_cnn_1.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(BinaryPretrainedCNNDataloader) // batch_size, eta_min=1e-6)

train_model(binary_cnn_1, BinaryPretrainedCNNDataloader, criterion, optimizer, lr_scheduler, num_epochs=n
```

```
um_epochs)
```

```
Training:   3%|▍          | 1/30 [00:22<10:55, 22.60s/it]
Epoch 1, loss: 0.3904079989064485
Training:  37%|███▋       | 11/30 [03:36<06:09, 19.44s/it]
Epoch 11, loss: 0.1769079332705587
Training:  70%|███████    | 21/30 [06:48<02:50, 18.98s/it]
Epoch 21, loss: 0.14551416088361294
Training: 100%|██████████| 30/30 [09:43<00:00, 19.45s/it]
```

In [238]:

```python
BinaryPretrainedCNNDataloader_test = DataLoader(ReviewsDataset(reviews_test[labels_test != 3], labels_tes
t[labels_test != 3], wv, max_seq_len=50, average_pooling=False),
                                                    batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(binary_cnn_1, BinaryPretrainedCNNDataloader_test))
```

```
Testing: 100%|██████████| 80/80 [00:13<00:00,  6.04it/s]
Classification Accuracy: 0.919525
```

Trained Word2Vec

In [245]:

```python
num_epochs = 20
batch_size = 500

BinaryTrainedCNNDataset = ReviewsDataset(reviews_train[labels_train != 3], labels_train[labels_train != 3
], model.wv, max_seq_len=50, average_pooling=False)
BinaryTrainedCNNDataloader = DataLoader(BinaryTrainedCNNDataset, batch_size=batch_size, shuffle=True, num
_workers=24)

input_dim = BinaryTrainedCNNDataset[0][0].shape[1]

binary_cnn_2 = CNN(input_dim, 2)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
binary_cnn_2 = binary_cnn_2.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(binary_cnn_2.parameters(), lr=0.001)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(BinaryTrainedCN
NDataloader) // batch_size, eta_min=1e-6)

train_model(binary_cnn_2, BinaryTrainedCNNDataloader, criterion, optimizer, lr_scheduler, num_epochs=num_
epochs)
```

```
Training:   0%|          | 0/20 [00:00<?, ?it/s]Training:   5%|▌         | 1/20 [00:20<06:20, 20.03s/it]
Epoch 1, loss: 0.2381597325205803
Training:  55%|█████▌    | 11/20 [03:37<02:56, 19.66s/it]
Epoch 11, loss: 0.10566438717069104
Training: 100%|██████████| 20/20 [06:37<00:00, 19.86s/it]
```

In [246]:

```python
BinaryTrainedCNNDataloader_test = DataLoader(ReviewsDataset(reviews_test[labels_test != 3], labels_test[l
abels_test != 3], model.wv, max_seq_len=50, average_pooling=False),
                                                    batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(binary_cnn_2, BinaryTrainedCNNDataloader_test))
```

```
Testing:   0%|          | 0/80 [00:00<?, ?it/s]Testing: 100%|██████████| 80/80 [00:12<00:00,  6.36it/s]
Classification Accuracy: 0.9204
```

Ternary Classification

Pretrained Word2Vec

In [280]:

```python
num_epochs = 30
```

```
num_epochs = 30
batch_size = 500

TernaryPretrainedCNNDataset = ReviewsDataset(reviews_train, labels_train, wv, max_seq_len=50, average_pooling=False)
TernaryPretrainedCNNDataloader = DataLoader(TernaryPretrainedCNNDataset, batch_size=batch_size, shuffle=True, num_workers=24)

input_dim = TernaryPretrainedCNNDataset[0][0].shape[1]

ternary_cnn_1 = CNN(input_dim, 3)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ternary_cnn_1 = ternary_cnn_1.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(ternary_cnn_1.parameters(), lr=0.01)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(TernaryPretrainedCNNDataloader) // batch_size, eta_min=1e-6)

train_model(ternary_cnn_1, TernaryPretrainedCNNDataloader, criterion, optimizer, lr_scheduler, num_epochs=num_epochs)
```

```
Training:   0%|          | 0/30 [00:00<?, ?it/s]Training:   3%|▎         | 1/30 [00:22<10:51, 22.46s/it]
Epoch 1, loss: 0.5850946374237538
Training:  37%|███▋      | 11/30 [04:04<06:58, 22.03s/it]
Epoch 11, loss: 0.3853776513040066
Training:  70%|███████   | 21/30 [07:43<03:17, 21.95s/it]
Epoch 21, loss: 0.33636606313288214
Training: 100%|██████████| 30/30 [11:01<00:00, 22.06s/it]
```

In [281]:

```
TernaryPretrainedCNNDataloader_test = DataLoader(ReviewsDataset(reviews_test, labels_test, wv, max_seq_len=50, average_pooling=False),
                                                 batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(ternary_cnn_1, TernaryPretrainedCNNDataloader_test))
```

```
Testing: 100%|██████████| 100/100 [00:13<00:00,  7.15it/s]
Classification Accuracy: 0.77694
```

Trained Word2Vec

In [282]:

```
num_epochs = 30
batch_size = 500

TernaryTrainedCNNDataset = ReviewsDataset(reviews_train, labels_train, model.wv, max_seq_len=50, average_pooling=False)
TernaryTrainedCNNDataloader = DataLoader(TernaryTrainedCNNDataset, batch_size=batch_size, shuffle=True, num_workers=24)

input_dim = TernaryTrainedCNNDataset[0][0].shape[1]

ternary_cnn_2 = CNN(input_dim, 3)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ternary_cnn_2 = ternary_cnn_2.to(device)
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = torch.optim.AdamW(ternary_cnn_2.parameters(), lr=0.01)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs*len(TernaryTrainedCNNDataloader) // batch_size, eta_min=1e-6)

train_model(ternary_cnn_2, TernaryTrainedCNNDataloader, criterion, optimizer, lr_scheduler, num_epochs=num_epochs)
```

```
Training:   3%|▎         | 1/30 [00:22<10:49, 22.39s/it]
Epoch 1, loss: 0.5495232558250427
Training:  37%|███▋      | 11/30 [04:05<07:04, 22.33s/it]
Epoch 11, loss: 0.41949798181653025
Training:  70%|███████   | 21/30 [07:45<03:18, 22.10s/it]
```

```
Epoch 21, loss: 0.38223037622869016
```

```
Training: 100%|██████████| 30/30 [11:05<00:00, 22.19s/it]
```

In [283]:

```
TernaryTrainedCNNDataloader_test = DataLoader(ReviewsDataset(reviews_test, labels_test, model.wv, max_seq
_len=50, average_pooling=False),
                                              batch_size=500, shuffle=False, num_workers=24)
print("Classification Accuracy:", test_model(ternary_cnn_2, TernaryTrainedCNNDataloader_test))
```

```
Testing: 100%|██████████| 100/100 [00:13<00:00,  7.41it/s]
```

```
Classification Accuracy: 0.78948
```