گزارش تمرین سوم

درس NLP

دانشکده مهندسی کامپیوتر

استاد درس: دکتر حمیدرضا برادران

دستیاران درس: خانم مظاهری و خانم کوپایی

سیاوش امیرحاجلو

993613007

خرداد 1403

مقدمه

این گزارش به جزئیات فرآیند ساخت یک مدل تحلیل احساسات با استفاده از شبکه‌های عصبی بازگشتی (RNN) می‌پردازد. این کار شامل پیش پردازش مجموعه داده بزرگی از توییت ها، بردار کردن داده های متنی، آموزش یک مدل RNN و ارزیابی عملکرد آن است. این گزارش تجزیه و تحلیلی از هر مرحله از فرآیند، از پیش پردازش داده ها تا ارزیابی مدل ارائه می دهد.

کتابخانه ها و محیط اجرای برنامه

```python
import pandas as pd
import numpy as np
import zipfile
import os
from collections import Counter
from itertools import chain

import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import torch
from torch.nn.utils.rnn import pad_sequence
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import torch.multiprocessing as mp

import gensim.downloader as api

# from google.colab import drive

import pickle

import matplotlib.pyplot as plt
```

کتابخانه های زیر برای انجام وظایف مختلف استفاده می شوند:

- pandas و numpy برای کار با داده ها و عملیات عددی.
- nltk برای عملیات پردازش زبان طبیعی.
- torch برای توسعه مدل یادگیری عمیق
- gensim برای embedding کلمات از پیش آموزش دیده.
- sklearn برای تقسیم داده ها و معیارهای ارزیابی داده.
- matplotlib برای رسم نمودار.

Data Loading

مجموعه داده مورد استفاده برای این کار مجموعه داده Sentiment140 است که شامل 1.6 میلیون توییت است که با برچسب‌های احساسات مثبت و منفی قرار داده شده‌اند. مجموعه داده به صورت تکه به تکه بارگذاری می‌شود تا استفاده از memory را به طور موثر مدیریت کند. به همین منظور، تنها نیمی از داده ها به صورت رندوم استفاده می‌شوند.

پیش پردازش داده ها

1. تبدیل برچسب‌ها

برچسب‌های احساسات به فرمت باینری تبدیل می‌شوند، جایی که توییت‌های مثبت به عنوان 1 و توییت‌های منفی 0 برچسب‌گذاری می‌شوند.

2. Text Cleaning

متن توییت‌ها برای جایگزینی URLها، منشن‌ها و هشتگ‌ها با توکن‌های خاص از پیش پردازش شده است. علائم نگارشی حذف می‌شود و متن نشانه‌گذاری می‌شود.

```
def preprocess_text(text):
    text = re.sub(r"http\S+|www\S+|https\S+", 'URL', text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+', 'MENTION', text)
    text = re.sub(r'\#\w+', 'HASHTAG', text)
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    return nltk.word_tokenize(text)
```

```
+ Code  + Markdown
```

```
def preprocess_chunk(chunk):
    chunk['text'] = chunk['text'].apply(preprocess_text)
    chunk['text'] = chunk['text'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])
    return chunk
```

```
# Apply preprocessing in chunks
processed_chunks = [preprocess_chunk(chunk) for chunk in np.array_split(df, len(df) // chunk_size)]
df = pd.concat(processed_chunks)
```

مراحل پیش پردازش تضمین می کند که داده های متنی در قالب مناسبی برای آموزش مدل هستند.

3. تقسیم داده ها

مجموعه داده به مجموعه های آموزشی (80٪) و آزمایشی (20٪) تقسیم می شود.

1-7 Consider 80% of the dataset for training and the remaining 20% for testing.

```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

وکتورسازی متن و Padding

1. ساخت Vocabulary

واژگانی از داده های آموزش ساخته می شود و به هر کلمه یک شاخص منحصر به فرد اختصاص می یابد.

```
# Build vocabulary
vocab = Counter(chain(*train_df['text']))
vocab = {word: idx + 1 for idx, (word, _) in enumerate(vocab.items())}
```

2. تبدیل توالی و padding

داده‌های متنی بر اساس واژگان به دنباله‌ای از شاخص‌ها تبدیل می‌شوند و دنباله‌ها برای اطمینان از طول یکنواخت قرار می‌گیرند.

- یک پیشنهاد جهت بهتر شدن نتیجه میتواند padding هم از قبل جمله و هم در بعد از جمله، به جای فقط قبل جمله باشد.

```python
# Function to convert words to indices
def text_to_sequence(text, vocab):
    return [vocab.get(word, 0) for word in text]
```

```python
# Dataset class
class TextDataset(Dataset):
    def __init__(self, df, vocab):
        self.texts = df['text'].apply(lambda x: text_to_sequence(x, vocab)).tolist()
        self.labels = df['sentiment'].tolist()

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        return torch.tensor(self.texts[idx], dtype=torch.long), torch.tensor(self.labels[idx], dtype=torch.float32)
```

```python
# Custom collate function to pad sequences within each batch
def collate_fn(batch):
    texts, labels = zip(*batch)
    texts_padded = pad_sequence(texts, batch_first=True, padding_value=0)
    labels = torch.tensor(labels, dtype=torch.float32)
    return texts_padded.to(device), labels.to(device)
```

```python
# Create Datasets
train_dataset = TextDataset(train_df, vocab)
test_dataset = TextDataset(test_df, vocab)

# Create DataLoaders with custom collate function
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, collate_fn=collate_fn, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, collate_fn=collate_fn, pin_memory=True)
```

Embedding کلمه

یک مدل Word2Vec از پیش آموزش دیده از Google News برای مقداردهی اولیه ماتریس Embedding استفاده می شود.

3) Word Embedding

```
# Load Google News Word2Vec model
w2v = api.load('word2vec-google-news-300')
```

```
# Initialize the embedding matrix
embedding_matrix = np.zeros((len(vocab) + 1, embedding_dim))

for word, idx in vocab.items():
    if word in w2v:
        embedding_matrix[idx] = w2v[word]
    else:
        # Initialize a random vector if the word is not in Word2Vec
        embedding_matrix[idx] = np.random.normal(size=(embedding_dim,))

# Convert embedding matrix to a tensor
embedding_matrix = torch.tensor(embedding_matrix, dtype=torch.float32).to(device)
```

تعریف و آموزش مدل

یک مدل RNN با یک لایه embedding، یک لایه RNN و یک لایه کاملا متصل برای classification تعریف می شود.

4) Define the RNN model, Training and Testing

```python
class RNNModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, embedding_matrix, hidden_dim, output_dim):
        super(RNNModel, self).__init__()
        self.embedding = nn.Embedding.from_pretrained(embedding_matrix, freeze=False)
        self.rnn = nn.RNN(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        embedded = self.embedding(x)
        output, hidden = self.rnn(embedded)
        hidden = hidden[-1]
        output = self.fc(hidden)
        return self.sigmoid(output)
```

4-1

```python
# Initialize the model, criterion, and optimizer
model = RNNModel(len(vocab) + 1, embedding_dim, embedding_matrix, hidden_dim, output_dim).to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters())

# Mixed precision training
scaler = torch.cuda.amp.GradScaler()
```

حلقه آموزش

این مدل برای 5 دوره آموزش داده می شود و error تمرین ثبت می شود.

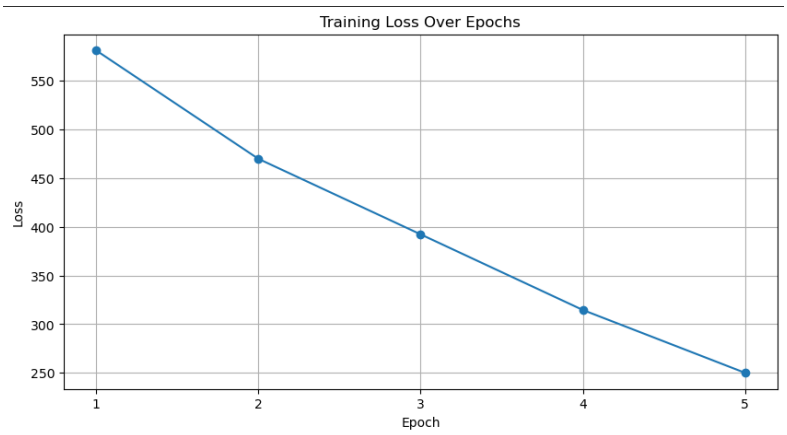
```python
train_losses = []

# Training loop
for epoch in range(epochs):
    model.train()
    epoch_loss = 0
    for texts, labels in train_loader:
        optimizer.zero_grad()
        with torch.cuda.amp.autocast():  # Mixed precision
            predictions = model(texts).squeeze()
            loss = criterion(predictions, labels)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        epoch_loss += loss.item()

    train_losses.append(epoch_loss)
    print(f'Epoch {epoch+1}, Loss: {epoch_loss/len(train_loader)}')
```

نتایج آموزش

نمودار تغییرات loss به شکل زیر است:



```
Epoch 1, Loss: 0.46487072269916535
Epoch 2, Loss: 0.3758363874435425
Epoch 3, Loss: 0.3138071233868599
Epoch 4, Loss: 0.25167330372333524
Epoch 5, Loss: 0.20004773244857788
```

ارزیابی مدل

مدل بر روی مجموعه تست ارزیابی می شود و یک گزارش classification تهیه می شود.

4-2 Evaluation

```
model.eval()
test_loss = 0
predictions = []
with torch.
    for batch in test_loader:
        texts, labels = batch
        preds = model(texts).squeeze()
        loss = criterion(preds, labels)
        test_loss += loss.item()
        predictions.extend((preds > 0.5).int().tolist())

print(f"Test Loss: {test_loss/len(test_loader)}")
```

(function) no_grad: Any

```
Test Loss: 0.5507221936989135
```

```
# Generate a classification report
test_labels_list = [label.item() for _, label in test_dataset]
print(classification_report(test_labels_list, predictions, target_names=['Negative', 'Positive']))
```

گزارش ارزیابی

	precision	recall	f1-score	support
Negative	0.78	0.83	0.81	79976
Positive	0.82	0.77	0.80	80024
accuracy			0.80	160000
macro avg	0.80	0.80	0.80	160000
weighted avg	0.80	0.80	0.80	160000

```
Test Loss: 0.5507221936989135
```