LAB 12.4 – 1D CNN with Pretrained GloVe

2403A52006

MD.Mustafa

Install & Import Libraries

```
"""
Lab 12.4
Text Classification using 1D CNN with Pretrained GloVe
"""

# PyTorch
import torch
import torch.nn as nn
import torch.optim as optim

# Data handling
import numpy as np
import pandas as pd
import re
import string

# Utilities
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from collections import Counter

import seaborn as sns
import matplotlib.pyplot as plt
```

Load SMS Dataset

```
"""
Load SMSSpamCollection dataset.
Format: label \t message
"""

data = pd.read_csv("SMSSpamCollection", sep="\t", header=None)
data.columns = ["label", "text"]

data.head()
```

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Next steps: Generate code with data New interactive sheet

Preprocess Text

```
"""
Basic text cleaning:
- Lowercase
- Remove digits
- Remove punctuation
"""

def clean_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text
```

```
data["clean_text"] = data["text"].apply(clean_text)

# Convert labels
data["label"] = data["label"].map({"ham": 0, "spam": 1})

data.head()
```

	label	text	clean_text
0	0	Go until jurong point, crazy.. Available only ...	go until jurong point crazy available only in ...
1	0	Ok lar... Joking wif u oni...	ok lar joking wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	free entry in a wkly comp to win fa cup final...
3	0	U dun say so early hor... U c already then say...	u dun say so early hor u c already then say
4	0	Nah I don't think he goes to usf, he lives aro...	nah i dont think he goes to usf he lives aroun...

Next steps: Generate code with data New interactive sheet

Build Vocabulary

```
"""
Create vocabulary dictionary.
Ignore very rare words.
"""

all_words = " ".join(data["clean_text"]).split()
word_counts = Counter(all_words)

vocab = {word: i+2 for i, (word, count) in enumerate(word_counts.items()) if count >= 2}

vocab["<PAD>"] = 0
vocab["<UNK>"] = 1

vocab_size = len(vocab)
print("Vocabulary Size:", vocab_size)
```

```
Vocabulary Size: 4009
```

Convert Text to Sequences

```
"""
Convert text into fixed-length index sequences.
"""

max_len = 50

def text_to_sequence(text):
    tokens = text.split()
    seq = [vocab.get(word, vocab["<UNK>"]) for word in tokens]

    if len(seq) < max_len:
        seq += [vocab["<PAD>"]] * (max_len - len(seq))
    else:
        seq = seq[:max_len]

    return seq

data["sequence"] = data["clean_text"].apply(text_to_sequence)

X = np.array(data["sequence"].tolist())
y = np.array(data["label"].tolist())
```

Train-Test Split

```
"""
Split data into training and testing.
Use stratify to preserve class balance.
"""

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

X_train = torch.tensor(X_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.long)
y_train = torch.tensor(y_train, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)
```

Download and Load GloVe

```
Start coding or generate with AI.
```

```
"""
Download and load GloVe 100d embeddings.
"""

!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip

embedding_dim = 100
glove_dict = {}

with open("glove.6B.100d.txt", encoding="utf8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.array(values[1:], dtype="float32")
        glove_dict[word] = vector

print("Loaded GloVe Words:", len(glove_dict))
```

```
--2026-02-19 04:50:39--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2026-02-19 04:50:39--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2026-02-19 04:50:40--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip.1’

glove.6B.zip.1      100%[===================>] 822.24M  2.72MB/s    in 4m 8s

2026-02-19 04:54:48 (3.31 MB/s) - ‘glove.6B.zip.1’ saved [862182613/862182613]

Archive:  glove.6B.zip
replace glove.6B.50d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: a
error:  invalid response [a]
replace glove.6B.50d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt       A

Loaded GloVe Words: 400000
```

Create Embedding Matrix

```
"""
Create embedding matrix correctly.
Matrix size must be max index + 1.
"""

# Get maximum index in vocab
max_index = max(vocab.values())

# Create embedding matrix
embedding_matrix = np.zeros((max_index + 1, embedding_dim))
```

```
for word, index in vocab.items():
    vector = glove_dict.get(word)
    if vector is not None:
        embedding_matrix[index] = vector

embedding_matrix = torch.tensor(embedding_matrix, dtype=torch.float32)

print("Embedding matrix shape:", embedding_matrix.shape)
```

```
Embedding matrix shape: torch.Size([8585, 100])
```

Define CNN Model

```
"""
1D CNN model with trainable embeddings.
Uses BCEWithLogitsLoss (no sigmoid inside).
"""

class TextCNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, embedding_matrix):
        super(TextCNN, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.embedding.weight = nn.Parameter(embedding_matrix)
        self.embedding.weight.requires_grad = True

        self.conv = nn.Conv1d(
            in_channels=embedding_dim,
            out_channels=128,
            kernel_size=5
        )

        self.pool = nn.MaxPool1d(kernel_size=2)
        self.fc = nn.Linear(128 * 23, 1)

    def forward(self, x):
        x = self.embedding(x)
        x = x.permute(0, 2, 1)
        x = torch.relu(self.conv(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

Initialize Model

```
"""
Handle class imbalance using weighted loss.
"""

model = TextCNN(vocab_size, embedding_dim, embedding_matrix)

class_counts = np.bincount(y_train.numpy().astype(int))
total = sum(class_counts)

weight_for_spam = total / (2 * class_counts[1])

criterion = nn.BCEWithLogitsLoss(pos_weight=torch.tensor(weight_for_spam))
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Train Model

```
"""
Train model for 10 epochs.
"""

epochs = 10

for epoch in range(epochs):
    model.train()

    outputs = model(X_train)
    loss = criterion(outputs.squeeze(), y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f}")
```

```
Epoch 1/10, Loss: 0.9487
Epoch 2/10, Loss: 0.8825
Epoch 3/10, Loss: 0.8232
Epoch 4/10, Loss: 0.7645
Epoch 5/10, Loss: 0.7056
Epoch 6/10, Loss: 0.6484
Epoch 7/10, Loss: 0.5965
Epoch 8/10, Loss: 0.5506
Epoch 9/10, Loss: 0.5086
Epoch 10/10, Loss: 0.4701
```

Evaluate Model

```
"""
Evaluate model performance.
"""

model.eval()

with torch.no_grad():
    outputs = model(X_test)
    probs = torch.sigmoid(outputs).squeeze()
    preds = (probs > 0.5).int().numpy()

accuracy = accuracy_score(y_test.numpy(), preds)

print("Accuracy:", accuracy)
print("\nClassification Report:\n")
print(classification_report(y_test.numpy(), preds))
```

```
Accuracy: 0.8977578475336323

Classification Report:

              precision    recall  f1-score   support

         0.0       0.96      0.92      0.94       966
         1.0       0.59      0.77      0.67       149

    accuracy                           0.90      1115
   macro avg       0.78      0.84      0.80      1115
weighted avg       0.91      0.90      0.90      1115
```

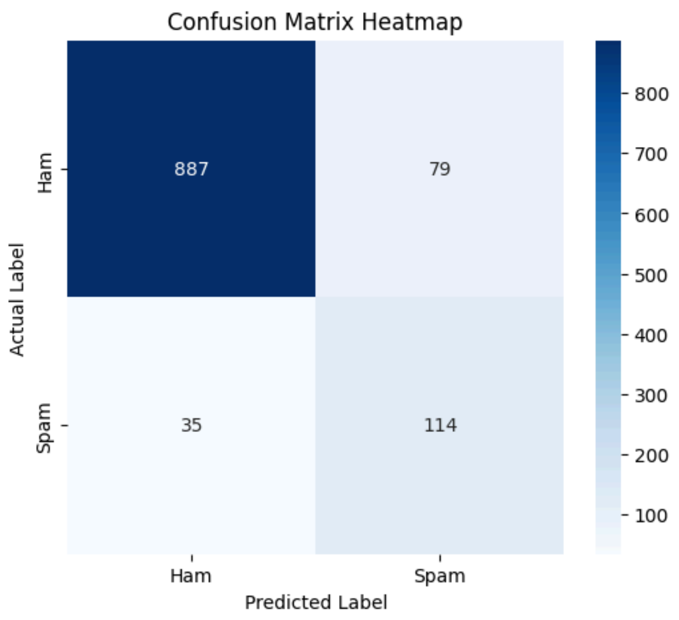
Confusion Matrix Heatmap

```
"""
Visualize confusion matrix.
"""

cm = confusion_matrix(y_test.numpy(), preds)

plt.figure(figsize=(6,5))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=["Ham", "Spam"],
    yticklabels=["Ham", "Spam"]
)

plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.show()
```


Confusion Matrix Heatmap
Actual Label
Predicted Label
Ham
Spam
Ham
Spam
887
79
35
114
800
700
600
500
400
300
200
100