```
import numpy as np
import pandas as pd
import re
import nltk
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
import matplotlib.pyplot as plt

# Download tokenizer resources
nltk.download('punkt')
from nltk.tokenize import word_tokenize
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
# Load SMS Spam dataset (tab-separated, no headers)
df = pd.read_csv("/content/SMSSpamCollection", sep="\t", header=None, names=["]

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

print(df.head())
print("Samples:", len(df))
print("Classes:", df['label'].unique())
```

```
   label                                             review
0      0  Go until jurong point, crazy.. Available only ...
1      0                      Ok lar... Joking wif u oni...
2      1  Free entry in 2 a wkly comp to win FA Cup fina...
3      0  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...
Samples: 5572
Classes: [0 1]
```

```
def preprocess(text):
    text = re.sub(r"[^a-zA-Z\s]", "", text.lower())
    tokens = word_tokenize(text)
    return tokens

nltk.download('punkt_tab') # Download 'punkt_tab' resource
df['tokens'] = df['review'].apply(preprocess)
print(df[['review','tokens']].head())
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
                                              review  \
0  Go until jurong point, crazy.. Available only ...
1                      Ok lar... Joking wif u oni...
2  Free entry in 2 a wkly comp to win FA Cup fina...
3  U dun say so early hor... U c already then say...
4  Nah I don't think he goes to usf, he lives aro...

                                              tokens
0  [go, until, jurong, point, crazy, available, o...
1                     [ok, lar, joking, wif, u, oni]
2  [free, entry, in, a, wkly, comp, to, win, fa, ...
3  [u, dun, say, so, early, hor, u, c, already, t...
4  [nah, i, dont, think, he, goes, to, usf, he, l...
```

```python
# Build vocabulary
word2idx = {"<pad>":0, "<unk>":1}
for tokens in df['tokens']:
    for token in tokens:
        if token not in word2idx:
            word2idx[token] = len(word2idx)

# Load GloVe embeddings (100d)
embedding_dim = 100
embeddings_index = {}

# Download GloVe embeddings if not already present
import os
if not os.path.exists("/content/glove.6B.100d.txt"):
    !wget http://nlp.stanford.edu/data/glove.6B.zip
    !unzip glove.6B.zip
    !mv glove.6B.100d.txt /content/

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

# Create embedding matrix
embedding_matrix = np.random.uniform(-0.25, 0.25, (len(word2idx), embedding_dim
for word, idx in word2idx.items():
    if word in embeddings_index:
        embedding_matrix[idx] = embeddings_index[word]
```

```
--2026-02-19 05:16:04--  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 05:16:04--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connecte
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2026-02-19 05:16:04--  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
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  4.89MB/s    in 2m 42s

2026-02-19 05:18:46 (5.08 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
mv: 'glove.6B.100d.txt' and '/content/glove.6B.100d.txt' are the same file
```

Gemini

```
# Convert tokens to numerical sequences
def tokens_to_indices(tokens, word2idx):
    return [word2idx.get(token, word2idx["<unk>"]) for token in tokens]

df['indices'] = df['tokens'].apply(lambda x: tokens_to_indices(x, word2idx))

# Pad sequences
max_len = max(len(x) for x in df['indices'])
padded_sequences = pad_sequence([torch.tensor(x) for x in df['indices']],
                                 batch_first=True,
                                 padding_value=word2idx["<pad>"])

# Convert labels to tensors
labels = torch.tensor(df['label'].values)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels

print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: torch.Size([4457, 171])
Shape of y_train: torch.Size([4457])
Shape of X_test: torch.Size([1115, 171])
Shape of y_test: torch.Size([1115])
```

```
train_texts, test_texts, train_labels, test_labels = train_test_split(
    df['tokens'], df['label'], test_size=0.2, random_state=42
)
```

```
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_classes, pretrained_weights):
        super(TextCNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.embedding.weight.data.copy_(torch.tensor(pretrained_weights))
        self.embedding.weight.requires_grad = False  # freeze embeddings

        self.conv1 = nn.Conv1d(embed_dim, 128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool1d(2)
        self.fc = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.embedding(x).permute(0, 2, 1)  # (batch, embed_dim, seq_len)
        x = self.pool(torch.relu(self.conv1(x)))
        x = torch.mean(x, dim=2)  # global pooling
        return self.fc(x)
```

```
class TextDataset(Dataset):
    def __init__(self, texts, labels, word2idx):
        self.texts = texts
        self.labels = labels
        self.word2idx = word2idx

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

    def __getitem__(self, idx):
        tokens = self.texts.iloc[idx]
        indices = [self.word2idx.get(token, self.word2idx["<unk>"]) for token i
        return torch.tensor(indices), torch.tensor(self.labels.iloc[idx])

def collate_fn(batch):
    texts, labels = zip(*batch)
    texts_padded = pad_sequence(texts, batch_first=True, padding_value=0)
    return texts_padded, torch.tensor(labels)

train_dataset = TextDataset(train_texts, train_labels, word2idx)
train_loader = DataLoader(train_dataset, batch_size=32, collate_fn=collate_fn)

model = TextCNN(len(word2idx), embedding_dim, 2, embedding_matrix)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

losses = []
for epoch in range(5):
    model.train()
    total_loss = 0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    losses.append(total_loss)
    print(f"Epoch {epoch+1}, Loss: {total_loss:.4f}")
```

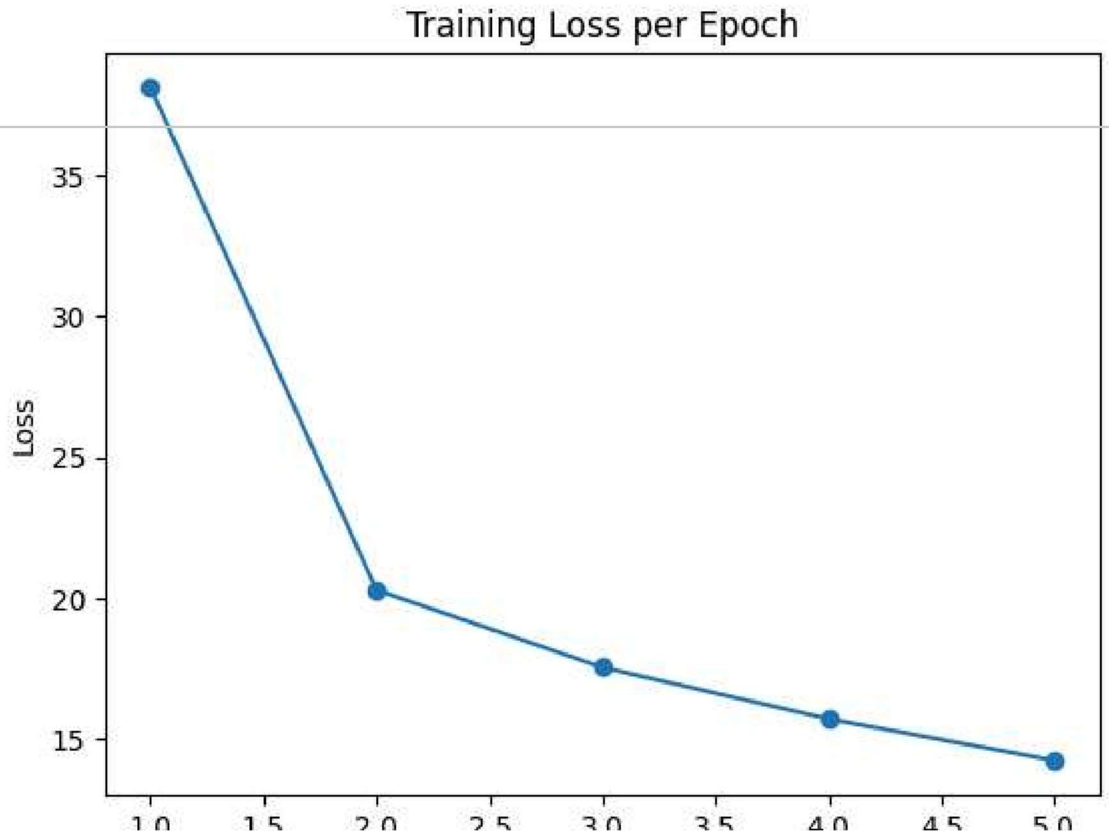
```
Epoch 1, Loss: 38.1482
Epoch 2, Loss: 20.2801
Epoch 3, Loss: 17.5324
Epoch 4, Loss: 15.7023
Epoch 5, Loss: 14.2270
```

```
test_dataset = TextDataset(test_texts, test_labels, word2idx)
test_loader = DataLoader(test_dataset, batch_size=32, collate_fn=collate_fn)

model.eval()
y_pred, y_true = [], []
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        preds = torch.argmax(outputs, dim=1)
        y_pred.extend(preds.tolist())
        y_true.extend(labels.tolist())

print("Accuracy:", accuracy_score(y_true, y_pred))
print("Precision:", precision_score(y_true, y_pred))
print("Recall:", recall_score(y_true, y_pred))
print("F1:", f1_score(y_true, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
```

```
Accuracy: 0.9587443946188341
Precision: 0.9401709401709402
Recall: 0.738255033557047
F1: 0.8270676691729323
Confusion Matrix:
 [[959   7]
 [ 39 110]]
```

```
# Plot training loss
plt.plot(range(1, len(losses)+1), losses, marker='o')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss per Epoch")
plt.show()

# Confusion matrix heatmap
```

```
import seaborn as sns
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Ham","Spam"],
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



Training Loss per Epoch
Loss
35
30
25
20
15
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0