Classification_of_textual_documents_p5

October 26, 2025

Desing and implement a deep learning network for classification of textual documents.

[9]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np

# Step 1: Load dataset
print("Step 1: Loading dataset ...")
data = fetch_20newsgroups(subset='all')
texts, labels = data.data, data.target
num_classes = len(set(labels))
print(f"Loaded {len(texts)} documents across {num_classes} classes.")

# Step 2: Tokenize & vectorize text
print("\nStep 2: Preprocessing text ...")
MAX_WORDS = 10000   # vocabulary size
MAX_LEN = 300       # max tokens per document

vectorizer = CountVectorizer(max_features=MAX_WORDS, stop_words='english')
X = vectorizer.fit_transform(texts).toarray()

# Pad/truncate to fixed length
if X.shape[1] < MAX_LEN:
    pad_width = MAX_LEN - X.shape[1]
    X = np.pad(X, ((0,0),(0,pad_width)), mode='constant')
else:
    X = X[:, :MAX_LEN]

# Convert to tensors
X = torch.tensor(X, dtype=torch.long)
y = torch.tensor(labels, dtype=torch.long)
```

```
# Step 3: Train-test split
print("\nStep 3: Splitting data ...")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64)

# Step 4: Build model
print("\nStep 4: Building deep learning model ...")
class TextClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_classes):
        super(TextClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, num_classes)

    def forward(self, x):
        embeds = self.embedding(x)
        lstm_out, _ = self.lstm(embeds)
        out = lstm_out[:, -1, :]  # last hidden state
        return self.fc(out)

vocab_size = MAX_WORDS
embed_dim = 128
hidden_dim = 128

model = TextClassifier(vocab_size, embed_dim, hidden_dim, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

print(model)

# Step 5: Train model
print("\nStep 5: Training model ...")
for epoch in range(3):  # keep small to run fast
    model.train()
    total_loss = 0
    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1} | Loss: {total_loss/len(train_loader):.4f}")
```

```python
# Step 6: Evaluate model
print("\nStep 6: Evaluating model ...")
model.eval()
correct, total = 0, 0
with torch.no_grad():
    for batch_X, batch_y in test_loader:
        outputs = model(batch_X)
        _, predicted = torch.max(outputs, 1)
        total += batch_y.size(0)
        correct += (predicted == batch_y).sum().item()

accuracy = correct / total
print(f"Test Accuracy: {accuracy:.4f}")
```

```
Step 1: Loading dataset …
Loaded 18846 documents across 20 classes.

Step 2: Preprocessing text …

Step 3: Splitting data …

Step 4: Building deep learning model …
TextClassifier(
  (embedding): Embedding(10000, 128)
  (lstm): LSTM(128, 128, batch_first=True)
  (fc): Linear(in_features=128, out_features=20, bias=True)
)

Step 5: Training model …
Epoch 1 | Loss: 2.9951
Epoch 2 | Loss: 2.9917
Epoch 3 | Loss: 2.9885

Step 6: Evaluating model …
Test Accuracy: 0.0626
```

[ ]: