Neural Networks

Author: Alikhan Semembayev

1. Perform necessary data preprocessing, e.g. removing punctuation and stop words, stemming, lemmatizing. You may use the outputs from previous weekly assignments. (10 points)

In [ ]:
```
from collections import defaultdict
import demoji
import svgling
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from autocorrect import Speller
import re

# Initialize tools
spell = Speller()
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

email_re = r"\b[A-Za-z]+@\S+\b"
ssn_re = r"\b[0-9]{3}-[0-9]{2}-[0-9]{4}\b"
ip_re = r"\b\d{1,3}[.]\d{1,3}[.]\d{1,3}[.]\d{1,3}\b"

street_number_re = r"^\d{1,}"
street_name_re = r"[a-zA-Z0-9\s]+,?"
city_name_re = r" [a-zA-Z]+(\,)?"
state_abbrev_re = r" [A-Z]{2}"
postal_code_re = r" [0-9]{5}$"
address_pattern_re = r"" + street_number_re + street_name_re + city_name_re + st


def clean_text(text):
    # Replace emojis
    text = demoji.replace(text)

    # Remove smart quotes and dashes
    text = text.replace("“", "\"").replace("”", "\"").replace("-", " ").replace(

    # Lowercase text
    text = text.lower()

    # Tokenize text
    words = word_tokenize(text)
    # print(words)

    # Spelling correction + replace all t with not
    words = ['not' if word == 't' else (
```

```
            'ADDRESS' if re.match(address_pattern_re, word)
            else (
                'EMAIL' if re.match(email_re, word)
                else (
                    'SSN' if re.match(ssn_re, word)
                    else (
                        'IP' if re.match(ip_re, word)
                        else spell(word)
                    )
                )
            )
    ) for word in words]

    # Remove stop words and non-alphabetic tokens and punctuation
    words = [word for word in words if word.isalnum() and word not in stop_words

    # POS tagging and Lemmatization
    tagged_words = pos_tag(words)

    tag_map = defaultdict(lambda: "n")
    tag_map["N"] = "n"
    tag_map["V"] = "v"
    tag_map["J"] = "a"
    tag_map["R"] = "r"

    words = [lemmatizer.lemmatize(word, pos=tag_map[tag[0]]) for word, tag in ta

    # Return cleaned words as a single string
    return ' '.join(words)
```

In [ ]:

```
import pandas as pd

data = (pd.read_csv('../../../../data/text/combined_raw.csv'))
data = data.dropna(how='any')

for row in data.values:
    row[0] = clean_text(row[0])

data.to_csv('../../../../data/text/combined_cleaned.csv', index=False)
```

In [6]:

```
import pandas as pd

data = (pd.read_csv('../../../../data/text/combined_cleaned.csv'))
data = data.dropna(how='any')

print(data.head(10))
```

```
                                              text    emotion
0  freshwater fish drink water skin via osmosis s...      happy
1  think everyone must use daily become grained e...    neutral
2  agree google headquarters mountain view califo...    neutral
3  thats funny current ceo sunday ficha didnt kno...    neutral
4  oh yeah not know either also want go google al...  surprised
5                                                say  surprised
6        yeah apparently lol instead hire people row      happy
7  thats funny guess imaginative leave huge tech ...  surprised
8  yeah exactly sure cheap one thing bet not expl...  surprised
9  remember hearing immortality waste jellyfish h...    neutral
```

2. For the binary classification problem you came up last week, set up a MLP to solve it. (50 points)

In [1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from transformers import BertTokenizer
import pandas as pd

# Custom Dataset Class
class EmotionDataset(Dataset):
    def __init__(self, texts, labels):
        self.texts = texts
        self.labels = labels

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

    def __getitem__(self, idx):
        return {
            'input_ids': self.texts[idx],
            'labels': self.labels[idx]
        }

# Define the Model
class EmotionClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_classes):
        super(EmotionClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.fc1 = nn.Linear(embed_dim * max_length, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, num_classes)
        self.dropout = nn.Dropout(0.5)

    def forward(self, input_ids):
        embedded = self.embedding(input_ids)
        embedded = embedded.view(embedded.size(0), -1)  # Flatten the embeddings
        x = torch.relu(self.fc1(embedded))
        x = self.dropout(x)
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In [2]:
```
import pandas as pd

data = (pd.read_csv('../../../../data/text/combined_cleaned.csv'))
data = data.dropna(how='any')

print(data.head(10))
```

```
                                                text    emotion
0  freshwater fish drink water skin via osmosis s...      happy
1  think everyone must use daily become grained e...    neutral
2  agree google headquarters mountain view califo...    neutral
3  thats funny current ceo sunday ficha didnt kno...    neutral
4  oh yeah not know either also want go google al...  surprised
5                                                say  surprised
6        yeah apparently lol instead hire people row      happy
7  thats funny guess imaginative leave huge tech ...  surprised
8  yeah exactly sure cheap one thing bet not expl...  surprised
9  remember hearing immortality waste jellyfish h...    neutral
```

In [4]:

```python
# Preprocess labels
label_encoder = LabelEncoder()
data['label'] = label_encoder.fit_transform(data['emotion'])
num_classes = len(label_encoder.classes_)

# Tokenization
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
max_length = 50

def tokenize_texts(texts):
    encodings = tokenizer(
        list(texts),
        padding='max_length',
        truncation=True,
        max_length=max_length,
        return_tensors='pt'
    )
    return encodings['input_ids'].tolist()

# Convert texts to token IDs
data['input_ids'] = tokenize_texts(data['text'])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    data['input_ids'].tolist(), data['label'].tolist(), test_size=0.2, random_st
)

# Convert data to PyTorch tensors
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.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

In [5]:

```python
# Create PyTorch datasets
train_dataset = EmotionDataset(X_train, y_train)
test_dataset = EmotionDataset(X_test, y_test)

# DataLoader
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16)
```

In [7]:

```python
# Check for CUDA
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Initialize model, loss, and optimizer
vocab_size = tokenizer.vocab_size  # Use tokenizer's vocab size
```

```
embed_dim = 50
model = EmotionClassifier(vocab_size, embed_dim, num_classes=num_classes).to(dev
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0004)

# Training loop
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    for batch in train_loader:
        optimizer.zero_grad()

        # Move data to device
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)

        # Forward pass
        outputs = model(input_ids)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

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

# Evaluation
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for batch in test_loader:
        # Move data to device
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids)
        preds = torch.argmax(outputs, dim=1)

        # Move predictions and labels back to CPU for evaluation
        all_preds.extend(preds.cpu().tolist())
        all_labels.extend(labels.cpu().tolist())

# Classification Report
print("Classification Report:")
print(classification_report(all_labels, all_preds, target_names=label_encoder.cl
print("Accuracy:", accuracy_score(all_labels, all_preds))
```

```
Using device: cuda
Epoch 1/10, Loss: 1.737734079360962
Epoch 2/10, Loss: 1.3603670597076416
Epoch 3/10, Loss: 1.4432244300842285
Epoch 4/10, Loss: 1.255486011505127
Epoch 5/10, Loss: 1.4124947786331177
Epoch 6/10, Loss: 1.295527458190918
Epoch 7/10, Loss: 1.4154021739959717
Epoch 8/10, Loss: 1.5953384637832642
Epoch 9/10, Loss: 1.2087442874908447
Epoch 10/10, Loss: 1.2303894758224487
Classification Report:
              precision    recall  f1-score   support

       angry       0.38      0.20      0.26      1518
     disgust       0.00      0.00      0.00       462
        fear       0.45      0.25      0.32      1631
       happy       0.46      0.42      0.44      8580
     neutral       0.45      0.54      0.49      8571
         sad       0.29      0.21      0.24      1864
   surprised       0.45      0.55      0.49      6850

    accuracy                           0.44     29476
   macro avg       0.35      0.31      0.32     29476
weighted avg       0.43      0.44      0.43     29476

Accuracy: 0.4440561813000407
```

3. Try to improve performance by modifying hyperparameters. (30 points)

In [11]:

```
# Check for CUDA
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Initialize model, loss, and optimizer
vocab_size = tokenizer.vocab_size  # Use tokenizer's vocab size
embed_dim = 50
model = EmotionClassifier(vocab_size, embed_dim, num_classes=num_classes).to(dev
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Training loop
num_epochs = 15
for epoch in range(num_epochs):
    model.train()
    for batch in train_loader:
        optimizer.zero_grad()

        # Move data to device
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)

        # Forward pass
        outputs = model(input_ids)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
```

```python
        loss.backward()
        optimizer.step()

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

# Evaluation
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for batch in test_loader:
        # Move data to device
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids)
        preds = torch.argmax(outputs, dim=1)

        # Move predictions and labels back to CPU for evaluation
        all_preds.extend(preds.cpu().tolist())
        all_labels.extend(labels.cpu().tolist())

# Classification Report
print("Classification Report:")
print(classification_report(all_labels, all_preds, target_names=label_encoder.cl
print("Accuracy:", accuracy_score(all_labels, all_preds))
```

```
Using device: cuda
Epoch 1/15, Loss: 1.5425374507904053
Epoch 2/15, Loss: 1.3963515758514404
Epoch 3/15, Loss: 1.3071774244308472
Epoch 4/15, Loss: 1.2933820486068726
Epoch 5/15, Loss: 1.2624475955963135
Epoch 6/15, Loss: 1.4152458906173706
Epoch 7/15, Loss: 1.472933292388916
Epoch 8/15, Loss: 1.231341004371643
Epoch 9/15, Loss: 1.1810821294784546
Epoch 10/15, Loss: 0.9682900309562683
Epoch 11/15, Loss: 0.9055383205413818
Epoch 12/15, Loss: 1.3185577392578125
Epoch 13/15, Loss: 1.6052885055541992
Epoch 14/15, Loss: 1.0228430032730103
Epoch 15/15, Loss: 1.3833003044128418
Classification Report:
              precision    recall  f1-score   support

       angry       0.45      0.06      0.11      1518
     disgust       0.00      0.00      0.00       462
        fear       0.37      0.47      0.41      1631
       happy       0.49      0.39      0.43      8580
     neutral       0.47      0.62      0.53      8571
         sad       0.33      0.20      0.25      1864
   surprised       0.46      0.53      0.49      6850

    accuracy                           0.46     29476
   macro avg       0.37      0.32      0.32     29476
weighted avg       0.45      0.46      0.44     29476

Accuracy: 0.45908535757904734
```

4. Summarize what you have learned and discovered from Task 1-3 as well as the tasks you completed last week.(10 points)

Summary of Findings

1. Preprocessing is very important stage of development. Using stemming and lemmatization, along with removing stop words, helps improve text data representation. It decreases the size of dataset by removing unnecessary information, and optimizes it for training.
2. Using an initial MLP setup, we achieved an accuracy of 44.4% on the binary classification problem. This MLP is very simple and has only 3 linear layers.
3. After tuning hyperparameters such as learning rate(0.0004=>0.0001) and number of epochs(10=>15), we noticed an improvement in accuracy to 45.9%. I believe that increasing the number of epochs will improve accuracy even more.
4. Binary classification has better accuracy, because of fewer labels(possible outputs). Multicalss problem is more difficult and requires bigger model and more time.