Name : `Rohan Ingle`

PRN: `22070126047`

Branch: `AIML A2 2022-2026`

GitHub Link : https://github.com/Rohan-ingle/Natural-Language-Processing

```
# !pip install pandas nltk scikit-learn rouge -qq
```

Importing Necessary Libraries

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from tqdm import tqdm
from rouge import Rouge
import os
from collections import Counter
import nltk
from nltk.tokenize import word_tokenize
#nltk.download('punkt', quiet=True)
#nltk.download('punkt_tab', quiet=True)
```

Defining Model Architecture

```
class BiLSTMSummarizer(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim,
output_dim):
        super(BiLSTMSummarizer, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.encoder = nn.LSTM(embedding_dim, hidden_dim,
bidirectional=True, batch_first=True)
        self.decoder = nn.LSTM(embedding_dim, hidden_dim * 2,
batch_first=True)
        self.fc = nn.Linear(hidden_dim * 2, output_dim)

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        batch_size = src.shape[0]
        trg_len = trg.shape[1]
        trg_vocab_size = self.fc.out_features
```

```
        outputs = torch.zeros(batch_size, trg_len,
trg_vocab_size).to(src.device)

        embedded = self.embedding(src)
        enc_output, (hidden, cell) = self.encoder(embedded)

        hidden = torch.cat((hidden[-2,:,:], hidden[-1,:,:]),
dim=1).unsqueeze(0)
        cell = torch.cat((cell[-2,:,:], cell[-1,:,:]),
dim=1).unsqueeze(0)

        input = trg[:, 0]

        for t in range(1, trg_len):
            input_embedded = self.embedding(input).unsqueeze(1)
            output, (hidden, cell) = self.decoder(input_embedded,
(hidden, cell))
            prediction = self.fc(output.squeeze(1))
            outputs[:, t] = prediction

            teacher_force = torch.rand(1).item() <
teacher_forcing_ratio
            top1 = prediction.argmax(1)
            input = trg[:, t] if teacher_force else top1

        return outputs
```

Making Functions to Parse Dataset, tokenize and build vocaboulary

```
def load_data(file_path):
    df = pd.read_csv(file_path)
    return df["Content"].tolist(), df["Headline"].tolist()

def tokenize(text):
    return word_tokenize(text.lower())

def build_vocab(texts, min_freq=2):
    word_freq = Counter()
    for text in texts:
        word_freq.update(text)

    vocab = {'<pad>': 0, '<unk>': 1, '<sos>': 2, '<eos>': 3}
    for word, freq in word_freq.items():
        if freq >= min_freq:
            vocab[word] = len(vocab)
```

```
    return vocab, {v: k for k, v in vocab.items()}
```

Loading The Data

```
articles, summaries =
load_data("/kaggle/input/hindi-summarization/hindi_news_dataset.csv")

tokenized_articles = [tokenize(article) for article in articles]
tokenized_summaries = [tokenize(summary) for summary in summaries]

vocab, inv_vocab = build_vocab(tokenized_articles +
tokenized_summaries)

train_articles, test_articles, train_summaries, test_summaries =
train_test_split(tokenized_articles, tokenized_summaries,
test_size=0.2, random_state=42)
train_articles, val_articles, train_summaries, val_summaries =
train_test_split(train_articles, train_summaries, test_size=0.1,
random_state=42)
```

Class to Initialize and preprocess the Dataset

```
class SummarizationDataset(Dataset):
    def __init__(self, articles, summaries, vocab, max_length=50):
        self.articles = articles
        self.summaries = summaries
        self.vocab = vocab
        self.max_length = max_length

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

    def __getitem__(self, idx):
        article = self.articles[idx]
        summary = self.summaries[idx]

        article_indices = [self.vocab['<sos>']] +
[self.vocab.get(token, self.vocab['<unk>']) for token in article]
[:self.max_length-2] + [self.vocab['<eos>']]
        summary_indices = [self.vocab['<sos>']] +
[self.vocab.get(token, self.vocab['<unk>']) for token in summary]
[:self.max_length-2] + [self.vocab['<eos>']]

        article_indices = article_indices + [self.vocab['<pad>']] *
(self.max_length - len(article_indices))
        summary_indices = summary_indices + [self.vocab['<pad>']] *
```

```
(self.max_length - len(summary_indices))

        return torch.tensor(article_indices),
torch.tensor(summary_indices)
```

Generating The Dataset fomr loaded data

```
train_dataset = SummarizationDataset(train_articles, train_summaries,
vocab)
val_dataset = SummarizationDataset(val_articles, val_summaries, vocab)
test_dataset = SummarizationDataset(test_articles, test_summaries,
vocab)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64)
test_loader = DataLoader(test_dataset, batch_size=64)
```

Defining Hyperparameters

```
vocab_size = len(vocab)
embedding_dim = 128
hidden_dim = 256
output_dim = vocab_size

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = BiLSTMSummarizer(vocab_size, embedding_dim, hidden_dim,
output_dim)
if torch.cuda.device_count() > 1:
    print(f"Using {torch.cuda.device_count()} GPUs")
    model = nn.DataParallel(model)

model = model.to(device)

Using 2 GPUs
```

Defining Training Function

```
def train(model, iterator, optimizer, criterion, device, clip=1,
teacher_forcing_ratio=0.5):
    model.train()
    epoch_loss = 0
    for batch in tqdm(iterator, desc="Training"):
        src, trg = batch
        src, trg = src.to(device), trg.to(device)
```

```
        optimizer.zero_grad()
        output = model(src, trg, teacher_forcing_ratio)

        output_dim = output.shape[-1]
        output = output[:, 1:].reshape(-1, output_dim)
        trg = trg[:, 1:].reshape(-1)

        loss = criterion(output, trg)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()

    return epoch_loss / len(iterator)
```

Defining Evaluation Function

```
def evaluate(model, iterator, criterion, device):
    model.eval()
    epoch_loss = 0
    with torch.no_grad():
        for batch in tqdm(iterator, desc="Evaluating"):
            src, trg = batch
            src, trg = src.to(device), trg.to(device)

            output = model(src, trg, 0)

            output_dim = output.shape[-1]
            output = output[:, 1:].reshape(-1, output_dim)
            trg = trg[:, 1:].reshape(-1)

            loss = criterion(output, trg)
            epoch_loss += loss.item()

    return epoch_loss / len(iterator)
```

Defining Beam Search Function

Beam search is a heuristic search algorithm that explores a graph by expanding the most promising nodes, maintaining a fixed number of best candidates (beam width) at each step to find the most likely sequence.

```
def beam_search(model, src, vocab, inv_vocab, beam_width=3,
max_length=50, min_length=10, device='cpu'):
    model.eval()
```

```
    if isinstance(model, nn.DataParallel):
        model = model.module

    with torch.no_grad():
        # Embedding the input sequence
        embedded = model.embedding(src)
        enc_output, (hidden, cell) = model.encoder(embedded)

        if model.encoder.bidirectional:
            hidden = torch.cat((hidden[-2, :, :], hidden[-1, :, :]),
dim=1)
            cell = torch.cat((cell[-2, :, :], cell[-1, :, :]), dim=1)
        else:
            hidden = hidden[-1, :, :]
            cell = cell[-1, :, :]

        hidden = hidden.unsqueeze(0)
        cell = cell.unsqueeze(0)

        beam = [([vocab['<sos>']], 0, hidden[:, 0:1, :], cell[:,
0:1, :])]
        complete_hypotheses = []

        for t in range(max_length):
            new_beam = []
            for seq, score, hidden, cell in beam:
                if seq[-1] == vocab['<eos>'] and len(seq) >=
min_length:
                    complete_hypotheses.append((seq, score))
                    continue

                input = torch.LongTensor([seq[-
1]]).unsqueeze(0).to(device)
                input_embedded = model.embedding(input)

                output, (hidden, cell) = model.decoder(input_embedded,
(hidden, cell))
                predictions = model.fc(output.squeeze(1))

                if len(seq) < min_length:
                    predictions[0][vocab['<eos>']] = float('-inf')

                top_preds = torch.topk(predictions, beam_width, dim=1)

                for i in range(beam_width):
                    new_seq = seq + [top_preds.indices[0][i].item()]
                    new_score = score - top_preds.values[0][i].item()
                    new_hidden = hidden.clone()
                    new_cell = cell.clone()
```

```
                    new_beam.append((new_seq, new_score, new_hidden, 
new_cell))

            beam = sorted(new_beam, key=lambda x: x[1])[:beam_width]

            if len(complete_hypotheses) >= beam_width:
                break

        complete_hypotheses = sorted(complete_hypotheses, key=lambda 
x: x[1])
        if complete_hypotheses:
            best_seq = complete_hypotheses[0][0]
        else:
            best_seq = beam[0][0]

    return [inv_vocab[idx] for idx in best_seq if idx not in 
[vocab['<sos>'], vocab['<eos>'], vocab['<pad>']]]
```

Defining a function to save model at specific instances

```
def save_model(model, vocab, filepath, embedding_dim, hidden_dim, 
output_dim):
    torch.save({
        'model_state_dict': model.state_dict(),
        'vocab': vocab,
        'embedding_dim': embedding_dim,
        'hidden_dim': hidden_dim,
        'output_dim': output_dim
    }, filepath)
    print(f"Model saved to {filepath}")
```

Training the model

```
optimizer = optim.Adam(model.parameters())
criterion = nn.CrossEntropyLoss(ignore_index=vocab['<pad>'])

num_epochs = 10
best_val_loss = float('inf')
for epoch in range(num_epochs):
    train_loss = train(model, train_loader, optimizer, criterion, 
device)
    val_loss = evaluate(model, val_loader, criterion, device)
    print(f'Epoch: {epoch+1:02}')
    print(f'\tTrain Loss: {train_loss:.3f}')
```

```
    print(f'\t Val. Loss: {val_loss:.3f}')

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        save_model(model, vocab, 'best_model.pth', embedding_dim,
hidden_dim, output_dim)
```

```
Training:   0%|          | 0/2087 [00:00<?,
?it/s]/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parall
el_apply.py:79: FutureWarning: `torch.cuda.amp.autocast(args...)` is
deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.device(device), torch.cuda.stream(stream),
autocast(enabled=autocast_enabled):
Training: 100%|██████████| 2087/2087 [34:00<00:00,  1.02it/s]
Evaluating: 100%|██████████| 232/232 [01:57<00:00,  1.98it/s]

Epoch: 01
	Train Loss: 5.241
	 Val. Loss: 4.580
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:52<00:00,  1.03it/s]
Evaluating: 100%|██████████| 232/232 [01:56<00:00,  1.99it/s]

Epoch: 02
	Train Loss: 2.947
	 Val. Loss: 3.413
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:55<00:00,  1.03it/s]
Evaluating: 100%|██████████| 232/232 [01:57<00:00,  1.98it/s]

Epoch: 03
	Train Loss: 1.996
	 Val. Loss: 2.777
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:55<00:00,  1.03it/s]
Evaluating: 100%|██████████| 232/232 [01:57<00:00,  1.98it/s]

Epoch: 04
	Train Loss: 1.491
	 Val. Loss: 2.415
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:57<00:00,  1.02it/s]
Evaluating: 100%|██████████| 232/232 [01:56<00:00,  1.98it/s]

Epoch: 05
	Train Loss: 1.173
	 Val. Loss: 2.154
Model saved to best_model.pth
```

```
Training: 100%|██████████| 2087/2087 [33:52<00:00,  1.03it/s]
Evaluating: 100%|██████████| 232/232 [01:57<00:00,  1.97it/s]

Epoch: 06
      Train Loss: 0.952
       Val. Loss: 1.961
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:58<00:00,  1.02it/s]
Evaluating: 100%|██████████| 232/232 [01:56<00:00,  1.98it/s]

Epoch: 07
      Train Loss: 0.798
       Val. Loss: 1.812
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:54<00:00,  1.03it/s]
Evaluating: 100%|██████████| 232/232 [01:56<00:00,  1.99it/s]

Epoch: 08
      Train Loss: 0.680
       Val. Loss: 1.696
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:59<00:00,  1.02it/s]
Evaluating: 100%|██████████| 232/232 [01:56<00:00,  1.98it/s]

Epoch: 09
      Train Loss: 0.592
       Val. Loss: 1.632
Model saved to best_model.pth

Training: 100%|██████████| 2087/2087 [33:59<00:00,  1.02it/s]
Evaluating: 100%|██████████| 232/232 [01:57<00:00,  1.98it/s]

Epoch: 10
      Train Loss: 0.519
       Val. Loss: 1.550
Model saved to best_model.pth
```

Defining a function to load model from checkpoints

```
def load_model(filepath, device):
    checkpoint = torch.load(filepath, map_location=device)
    vocab = checkpoint['vocab']

    embedding_dim = checkpoint['embedding_dim']
```

```
    hidden_dim = checkpoint['hidden_dim']
    output_dim = checkpoint['output_dim']
    vocab_size = len(vocab)

    model = BiLSTMSummarizer(vocab_size, embedding_dim, hidden_dim, 
output_dim).to(device)

    if torch.cuda.device_count() > 1:
        model = nn.DataParallel(model)

    # Load the model state
    model.load_state_dict(checkpoint['model_state_dict'])

    return model, vocab
```

Evaluation

```
best_model, _ = load_model('best_model.pth', device)

test_loss = evaluate(best_model, test_loader, criterion, device)
print(f'Test Loss: {test_loss:.3f}')

# Evaluate using ROUGE score
rouge = Rouge()
best_model.eval()
predictions = []
references = []
with torch.no_grad():
    for batch in tqdm(test_loader, desc="Generating summaries"):
        src, trg = batch
        src = src.to(device)
        pred = beam_search(best_model, src, vocab, inv_vocab, 
min_length=10, device=device)  # Set minimum length
        predictions.extend([' '.join(pred)])
        references.extend([' '.join([inv_vocab[idx.item()] for idx in 
trg[0] if idx.item() not in [vocab['<sos>'], vocab['<eos>'], 
vocab['<pad>']]])])

min_length = 10
predictions = [p if len(p.split()) >= min_length else p + ' ' + ' 
'.join(['<pad>'] * (min_length - len(p.split()))) for p in 
predictions]

scores = rouge.get_scores(predictions, references, avg=True)
print("ROUGE scores:")
print(scores)
```

```
/tmp/ipykernel_30/1194031576.py:3: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint = torch.load(filepath, map_location=device)
Evaluating: 100%|██████████| 580/580 [04:51<00:00,  1.99it/s]

Test Loss: 1.546

Generating summaries: 100%|██████████| 580/580 [00:38<00:00,
15.00it/s]

ROUGE scores:
{'rouge-1': {'r': 0.8848763375723716, 'p': 0.8715314577300806, 'f':
0.8749202841986269}, 'rouge-2': {'r': 0.8253247433147615, 'p':
0.7764323152900745, 'f': 0.7971380303678042}, 'rouge-l': {'r':
0.8734796275202089, 'p': 0.8588486369137195, 'f': 0.8630128449535202}}
```

```python
checkpoint = torch.load('best_model.pth', map_location=device)
print("Checkpoint Keys:", checkpoint.keys())
```

```
/tmp/ipykernel_30/4165049954.py:1: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint = torch.load('best_model.pth', map_location=device)

Checkpoint Keys: dict_keys(['model_state_dict', 'vocab',
'embedding_dim', 'hidden_dim', 'output_dim'])
```

```
print("Loading pre-trained model...")
trained_model, vocab = load_model('best_model.pth', device)
inv_vocab = {v: k for k, v in vocab.items()}
trained_model = trained_model.to(device)
```

```
Loading pre-trained model...
```

```
/tmp/ipykernel_30/1194031576.py:3: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  checkpoint = torch.load(filepath, map_location=device)
```

Getting summaries

```
def summarize_text(model, vocab, inv_vocab, text, max_length=50,
min_length=10, beam_width=3, device='cpu', debug=False):
    model.eval()
    tokens = tokenize(text)[:max_length]
    indices = [vocab['<sos>']] + [vocab.get(token, vocab['<unk>']) for
token in tokens] + [vocab['<eos>']]
    src = torch.LongTensor(indices).unsqueeze(0).to(device)

    summary = beam_search(model, src, vocab, inv_vocab, beam_width,
max_length, min_length, device)

    if debug:
        print("Input tokens:", tokens)
        print("Input indices:", indices)
        print("Generated indices:", [vocab[word] for word in summary])
        print("Summary length:", len(summary))

    return ' '.join(summary)

input_text = "ऑस्ट्रेलिया ने ब्लूमफोनटीन में पहले वनडे में दक्षिण अफ्रीका को 3-विकेट से हरा दिया।
यह 12 वर्षों में दक्षिण अफ्रीका के खिलाफ उसकी धरती पर ऑस्ट्रेलिया की पहली वनडे जीत है। ऑस्ट्रेलिया
का स्कोर 16.3 ओवर में 113/7 था लेकिन मार्नस लबुशेन और ऐश्टन एगर की 112* रनों की साझेदारी
```

```
की बदौलत उसने 40.2 ओवर में लक्ष्य हासिल कर लिया।"
summary = summarize_text(trained_model, vocab, inv_vocab, input_text,
min_length=10, device=device, debug=True)
print("Generated Summary:")
print(summary)
print("Summary length:", len(summary.split()))
```

```
Input tokens: ['ऑस्ट्रेलिया', 'ने', 'ब्लूमफोनटीन', 'में', 'पहले', 'वनडे', 'में',
'दक्षिण', 'अफ्रीका', 'को', '3-विकेट', 'से', 'हरा', 'दिया।', 'यह', '12', 'वर्षों',
'में', 'दक्षिण', 'अफ्रीका', 'के', 'खिलाफ', 'उसकी', 'धरती', 'पर', 'ऑस्ट्रेलिया',
'की', 'पहली', 'वनडे', 'जीत', 'है।', 'ऑस्ट्रेलिया', 'का', 'स्कोर', '16.3',
'ओवर', 'में', '113/7', 'था', 'लेकिन', 'मार्नस', 'लबुशेन', 'और', 'ऐश्टन',
'एगर', 'की', '112*', 'रनों', 'की', 'साझेदारी']
Input indices: [2, 4910, 37, 8412, 14, 193, 7443, 14, 2425, 2426, 10,
8413, 86, 7167, 96, 67, 1330, 1254, 14, 2425, 2426, 12, 189, 274,
8414, 50, 4910, 8, 725, 7443, 2940, 35, 4910, 71, 7770, 8415, 7479,
14, 8416, 428, 596, 7627, 7628, 43, 8417, 8418, 8, 8419, 7930, 8,
3882, 3]
Generated indices: [4910, 37, 1330, 1074, 14, 725, 2710, 2425, 2426,
10, 274, 8414, 50, 7443, 2524, 14, 7751, 50, 900]
Summary length: 19
Generated Summary:
ऑस्ट्रेलिया ने 12 साल में पहली बार दक्षिण अफ्रीका को उसकी धरती पर वनडे मैच में हराया पर पहुंचा
Summary length: 19
```