4 NLP Hindi_Summarization_Beam_Search

October 19, 2024

Name: Aparna Iyer

PRN: 22070126017

Batch: 2022-2026, AI-ML A1

[7]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
!pip install pandas nltk scikit-learn rouge -qq
```

[8]:
```
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)
```

[9]:
```
# Define the BiLSTM model
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
```

[10]:

```
# Custom dataset class
class SummarizationDataset(Dataset):
    def __init__(self, articles, summaries, vocab, max_length=100):
        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)
```

[20]:

```
# Load and preprocess data
def load_data(file_path):
    #df = pd.read_csv(file_path,nrows=10000)
    return df['Content'], df['Content'].tolist()
   # return df[C.tolist(Content)], df[Content].tolist()

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

# Build vocabulary
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()}
```

[24]:

```
!pip install nltk
import nltk

# Download the 'punkt' resource
nltk.download('punkt')
# Load data
file_path = '/content/drive/MyDrive/Colab Notebooks/Datasets/hindi_news_dataset.
↪csv';
articles, summaries = load_data(file_path)
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
```

```
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.5)

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

[25]:

```python
# Tokenize data
tokenized_articles = [tokenize(article) for article in articles]
tokenized_summaries = [tokenize(summary) for summary in summaries]
```

[27]:

```python
# Build vocabulary
vocab, inv_vocab = build_vocab(tokenized_articles + tokenized_summaries)
```

[28]:

```python
# Split data
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)
```

[30]:

```python
# Create datasets
'''train_dataset = SummarizationDataset(None, None, None)
val_dataset = SummarizationDataset(None, None, None)
test_dataset = SummarizationDataset(None, None, None)'''

train_dataset = SummarizationDataset(train_articles, train_summaries, vocab) # Pass train_articles, train_summaries, and vocab
val_dataset = SummarizationDataset(val_articles, val_summaries, vocab)      # Pass val_articles, val_summaries, and vocab
test_dataset = SummarizationDataset(test_articles, test_summaries, vocab)   # Pass test_articles, test_summaries, and vocab


# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=128)
test_loader = DataLoader(test_dataset, batch_size=128)
```

[32]:

```python
# Initialize model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
# Set appropriate values for vocab_size, embedding_dim, hidden_dim, and␣
↪output_dim
vocab_size = len(vocab)  # Assuming 'vocab' is your vocabulary dictionary
embedding_dim = 128  # Example value, adjust as needed
hidden_dim = 256    # Example value, adjust as needed
output_dim = vocab_size # Example value, adjust as needed
model = BiLSTMSummarizer(vocab_size, embedding_dim, hidden_dim, output_dim).
↪to(device) # Pass the device to the to() method
#model = BiLSTMSummarizer(None, None, None,None).to(None)
```

[33]:

```
# Train 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)
```

[34]:

```
# 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)  # turn off teacher forcing

            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)
```

[35]:

```
def beam_search(model, src, vocab, inv_vocab, beam_width=3, max_length=100,␣
↪min_length=10, device='cpu'):
    model.eval()
    with torch.no_grad():
        # Embedding the input sequence
        embedded = model.embedding(src)  # shape: (batch_size, seq_len,␣
↪embedding_dim)
        enc_output, (hidden, cell) = model.encoder(embedded)  # LSTM encoder␣
↪output

        # In case of bi-directional LSTM, combine the hidden states
        if model.encoder.bidirectional:
            hidden = torch.cat((hidden[-2, :, :], hidden[-1, :, :]), dim=1)  #␣
↪shape: (batch_size, hidden_dim)
            cell = torch.cat((cell[-2, :, :], cell[-1, :, :]), dim=1)        #␣
↪shape: (batch_size, hidden_dim)
        else:
            hidden = hidden[-1, :, :]  # Take the last layer if not␣
↪bi-directional
            cell = cell[-1, :, :]      # Take the last layer if not␣
↪bi-directional

        # Now we process one sequence at a time, so set batch size to 1
        hidden = hidden.unsqueeze(0)  # shape: (1, batch_size, hidden_dim)
        cell = cell.unsqueeze(0)      # shape: (1, batch_size, hidden_dim)

        # Initialize the beam with the start-of-sequence token
        beam = [([vocab['<sos>']], 0, hidden[:, 0:1, :], cell[:, 0:1, :])]  #␣
↪Start with one sequence
        complete_hypotheses = []

        # Perform beam search
        for t in range(max_length):
            new_beam = []
            for seq, score, hidden, cell in beam:
                # If end-of-sequence token is reached and length is >=␣
↪min_length, add to complete hypotheses
                if seq[-1] == vocab['<eos>'] and len(seq) >= min_length:
```

```
                complete_hypotheses.append((seq, score))
                continue

            # Prepare the input for the decoder (last predicted token)
            input = torch.LongTensor([seq[-1]]).unsqueeze(0).to(device)  # shape: (1, 1)
            input_embedded = model.embedding(input)  # shape: (1, 1, embedding_dim)

            # Pass through the decoder with the current hidden and cell states
            output, (hidden, cell) = model.decoder(input_embedded, (hidden, cell))  # hidden, cell are (1, 1, hidden_dim)
            predictions = model.fc(output.squeeze(1))  # shape: (1, vocab_size)

            # Prevent EOS if sequence is shorter than minimum length
            if len(seq) < min_length:
                predictions[0][vocab['<eos>']] = float('-inf')

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

            # For each top prediction, extend the sequence and update the beam
            for i in range(beam_width):
                new_seq = seq + [top_preds.indices[0][i].item()]
                new_score = score - top_preds.values[0][i].item()  # Negative log probability
                new_hidden = hidden.clone()
                new_cell = cell.clone()
                new_beam.append((new_seq, new_score, new_hidden, new_cell))

        # Sort by score and keep top beam_width sequences
        beam = sorted(new_beam, key=lambda x: x[1])[:beam_width]

        if len(complete_hypotheses) >= beam_width:
            break

    # Sort and return the best sequence
    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]
```

```
    # Convert sequence of indices back to words
    return [inv_vocab[idx] for idx in best_seq if idx not in [vocab['<sos>'],␣
↪vocab['<eos>'], vocab['<pad>']]]
```

[40]:
```
# Save model function
def save_model(model, vocab, filepath):
    torch.save({
        'model_state_dict': model.state_dict(),
        'vocab': vocab
    }, filepath)
    print(f"Model saved to {filepath}")
```

[41]:
```
# Define optimizer and loss function
optimizer = optim.Adam(model.parameters())
criterion = nn.CrossEntropyLoss(ignore_index=vocab['<pad>'])
```

[42]:
```
# Training loop
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}')

    # Debug: Check if val_loss and best_val_loss are valid and being compared␣
↪correctly
    print(f'Initial best_val_loss: {best_val_loss}')
    print(f'Epoch {epoch+1} val_loss: {val_loss}')

    # Save model if validation loss improves
    if val_loss < best_val_loss:
        print(f"New best val_loss: {val_loss} (Previous best: {best_val_loss})")
        best_val_loss = val_loss
        save_model(model, vocab, '/content/drive/MyDrive/Colab Notebooks/
↪Datasets/best_model.pth')
```

```
Training: 100%|          | 57/57 [01:17<00:00,  1.35s/it]
Evaluating: 100%|          | 7/7 [00:02<00:00,  2.35it/s]

Epoch: 01
        Train Loss: 4.804
         Val. Loss: 5.823
Initial best_val_loss: inf
Epoch 1 val_loss: 5.822606359209333
New best val_loss: 5.822606359209333 (Previous best: inf)
```

```
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:16<00:00,  1.35s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.48it/s]

Epoch: 02
        Train Loss: 4.477
         Val. Loss: 5.672
Initial best_val_loss: 5.822606359209333
Epoch 2 val_loss: 5.672139917101179
New best val_loss: 5.672139917101179 (Previous best: 5.822606359209333)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:16<00:00,  1.35s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.45it/s]

Epoch: 03
        Train Loss: 4.250
         Val. Loss: 5.576
Initial best_val_loss: 5.672139917101179
Epoch 3 val_loss: 5.575550488063267
New best val_loss: 5.575550488063267 (Previous best: 5.672139917101179)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:16<00:00,  1.35s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.46it/s]

Epoch: 04
        Train Loss: 4.099
         Val. Loss: 5.463
Initial best_val_loss: 5.575550488063267
Epoch 4 val_loss: 5.462789126804897
New best val_loss: 5.462789126804897 (Previous best: 5.575550488063267)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:17<00:00,  1.36s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.40it/s]

Epoch: 05
        Train Loss: 3.961
         Val. Loss: 5.384
Initial best_val_loss: 5.462789126804897
Epoch 5 val_loss: 5.383774893624442
New best val_loss: 5.383774893624442 (Previous best: 5.462789126804897)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:17<00:00,  1.36s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.43it/s]

Epoch: 06
        Train Loss: 3.848
         Val. Loss: 5.327
Initial best_val_loss: 5.383774893624442
```

```
Epoch 6 val_loss: 5.326958928789411
New best val_loss: 5.326958928789411 (Previous best: 5.383774893624442)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:17<00:00,  1.35s/it]
Evaluating: 100%|      | 7/7 [00:03<00:00,  2.32it/s]

Epoch: 07
        Train Loss: 3.771
         Val. Loss: 5.224
Initial best_val_loss: 5.326958928789411
Epoch 7 val_loss: 5.224493844168527
New best val_loss: 5.224493844168527 (Previous best: 5.326958928789411)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:17<00:00,  1.36s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.42it/s]

Epoch: 08
        Train Loss: 3.615
         Val. Loss: 5.177
Initial best_val_loss: 5.224493844168527
Epoch 8 val_loss: 5.177227292742048
New best val_loss: 5.177227292742048 (Previous best: 5.224493844168527)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:17<00:00,  1.36s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.43it/s]

Epoch: 09
        Train Loss: 3.521
         Val. Loss: 5.108
Initial best_val_loss: 5.177227292742048
Epoch 9 val_loss: 5.108450208391462
New best val_loss: 5.108450208391462 (Previous best: 5.177227292742048)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth

Training: 100%|      | 57/57 [01:17<00:00,  1.36s/it]
Evaluating: 100%|      | 7/7 [00:02<00:00,  2.37it/s]

Epoch: 10
        Train Loss: 3.378
         Val. Loss: 5.087
Initial best_val_loss: 5.108450208391462
Epoch 10 val_loss: 5.087335177830288
New best val_loss: 5.087335177830288 (Previous best: 5.108450208391462)
Model saved to /content/drive/MyDrive/Colab Notebooks/Datasets/best_model.pth
```

[50]:

```
# Load model function
def load_model(filepath, device):
    checkpoint = torch.load(filepath, map_location=device)
```

```
    vocab = checkpoint['vocab']

    # Get model parameters from the checkpoint or define them explicitly
    vocab_size = len(vocab) # Assuming vocab is a dictionary or list
    embedding_dim = checkpoint['model_state_dict']['embedding.weight'].shape[1]␣
↪ # Extract from checkpoint
    hidden_dim = checkpoint['model_state_dict']['encoder.weight_ih_l0'].
↪shape[0] // 4  # Extract from checkpoint, adjust for bidirectional
    #output_dim = checkpoint['model_state_dict']['decoder.weight_ih_l0'].
↪shape[0]  # Extract from
    output_dim = checkpoint['model_state_dict']['fc.weight'].shape[0]

    model = BiLSTMSummarizer(vocab_size, embedding_dim, hidden_dim, output_dim).
↪to(device)
    model.load_state_dict(checkpoint['model_state_dict'])
    return model, checkpoint
```

[51]:
```
# Load the best model for testing
best_model, _ = load_model('/content/drive/MyDrive/Colab Notebooks/Datasets/
↪best_model.pth', device)

# Test the model
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>']]])])

# Ensure all predictions meet the minimum length
min_length = 10  # Set this to your desired minimum length
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)
```

```python
print("ROUGE scores:")
print(scores)
```

```
<ipython-input-50-27d5bd466b8f>: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%|      | 16/16 [00:07<00:00,  2.22it/s]
```

```
Test Loss: 5.119
```

```
Generating summaries: 100%|      | 16/16 [00:02<00:00,  5.88it/s]
```

```
ROUGE scores:
{'rouge-1': {'r': 0.40538013045177357, 'p': 0.6120189874974145, 'f':
0.48564206461219067}, 'rouge-2': {'r': 0.19840822735777705, 'p':
0.24903439964140364, 'f': 0.22023135387366122}, 'rouge-l': {'r':
0.35727662611778066, 'p': 0.5409299400720472, 'f': 0.42847228528101516}}
```

[52]:
```python
print("Loading pre-trained model...")
trained_model, checkpoint = load_model('best_model.pth', device)
vocab = checkpoint['vocab']
inv_vocab = {v: k for k, v in vocab.items()}
trained_model = trained_model.to(device)
```

```
Loading pre-trained model…
```

```
<ipython-input-50-27d5bd466b8f>: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)
```

[53]:
```python
# Modified Summarization bot
def summarize_text(model, vocab, inv_vocab, text, max_length=100,␣
↪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)
```

[54]:
```python
# Example usage of the summarization bot
input_text = "                                    3-              12              ␣
↪                                   16.3      113/7                         ␣
↪112*                 40.2                 "
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: ['    ', ' ', '     ', ' ', '  ', '  ', ' ',
'   ', '   ', ' ', '3-  ', ' ', '  ', '   ', ' ', '12',
'  ', ' ', '   ', '    ', ' ', '  ', '  ', '  ', ' ',
'     ', ' ', '  ', '  ', '  ', ' ', '     ', ' ', '  ',
'16.3', '  ', ' ', '113/7', ' ', '  ', '   ', '  ', ' ', '  ',
'  ', ' ', '112', '*', '  ', ' ', '    ', ' ', '   ', '  ',
'40.2', '  ', ' ', '  ', '   ', ' ', '   ']
Input indices: [2, 4928, 37, 8392, 14, 193, 7433, 14, 2418, 2419, 10, 8393, 86,
7153, 96, 67, 1327, 1251, 14, 2418, 2419, 12, 189, 274, 8394, 50, 4928, 8, 723,
7433, 2952, 35, 4928, 71, 7759, 8395, 7469, 14, 8396, 428, 596, 7617, 7618, 43,
8397, 8398, 8, 8399, 7487, 7915, 8, 3914, 8, 7526, 434, 8400, 7469, 14, 4497,
4302, 20, 1955, 3]
Generated indices: [4928, 37, 8392, 14, 7433, 14, 14, 14, 10, 7433, 14, 14,
7433, 7535, 14, 7433, 14, 14, 237, 12, 189, 2516, 14, 237, 12, 189, 2516, 14,
```

```
237, 12, 189, 14, 14, 12, 12, 14, 14, 12, 14, 14, 14, 14, 14, 14, 8, 8, 8, 14,
14, 8, 8, 8, 8, 8, 8, 8, 35]
Summary length: 57
Generated Summary:



Summary length: 57
```

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

[60]:
```
!jupyter nbconvert --to PDF "/content/drive/MyDrive/Colab Notebooks/4 NLP␣
↪Hindi_Summarization_Beam_Search.ipynb"
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab Notebooks/4 NLP
Hindi_Summarization_Beam_Search.ipynb to PDF
[NbConvertApp] ERROR | Notebook JSON is invalid: Additional properties are not
allowed ('metadata' was unexpected)

Failed validating 'additionalProperties' in stream:

On instance['cells'][18]['outputs'][0]:
{'metadata': {'tags': None},
 'name': 'stderr',
 'output_type': 'stream',
 'text': 'Training: 100%|      | 57/57 [01:17<00:00,  1.35s/it]\n'
         'Evalua…'}
[NbConvertApp] Writing 84635 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 72377 bytes to /content/drive/MyDrive/Colab Notebooks/4
NLP Hindi_Summarization_Beam_Search.pdf
```