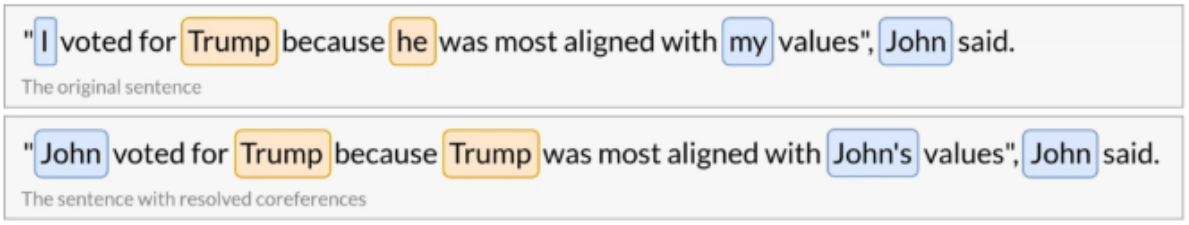
Experiment No. 8
Implement word sense disambiguation using LSTM/GRU
Date of Performance:
Date of Submission:

Aim: Apply Reference Resolution Technique on the given Text input.

Objective: Understand the importance of resolving references and implementing reference resolution for the given text input.

Theory:

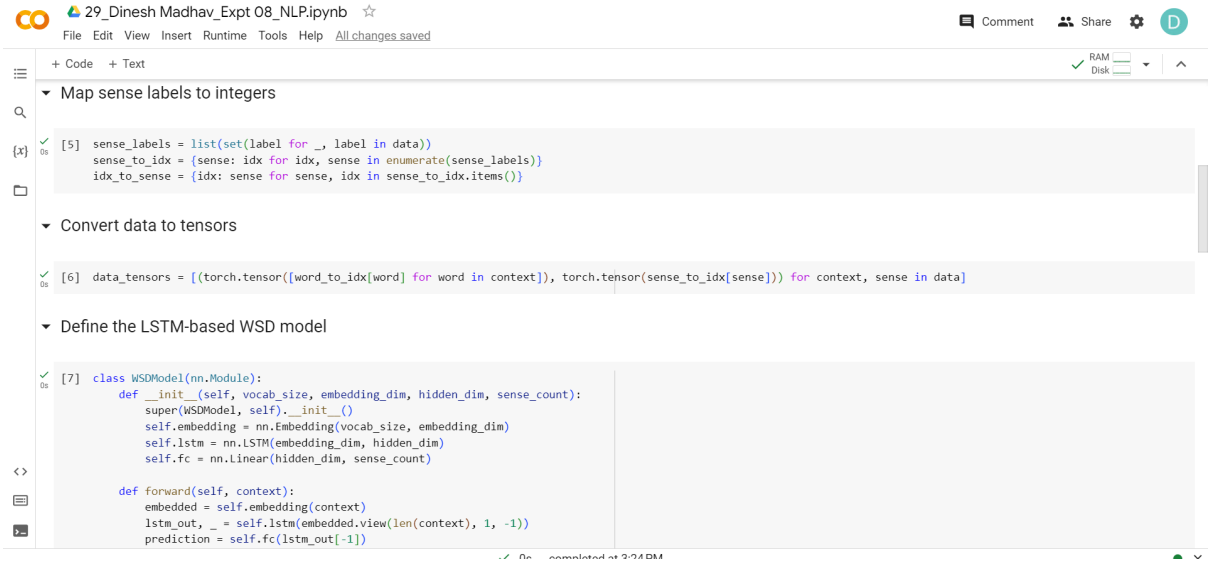
Coreference resolution (CR) is the task of finding all linguistic expressions (called mentions) in a given text that refer to the same real-world entity. After finding and grouping these mentions we can resolve them by replacing, as stated above, pronouns with noun phrases.

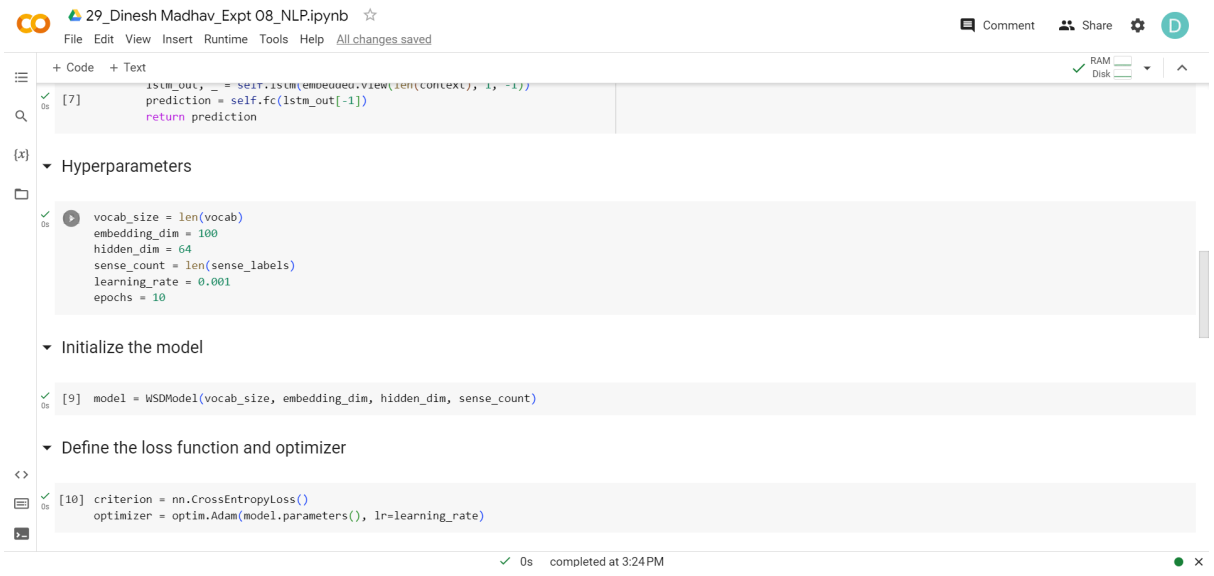


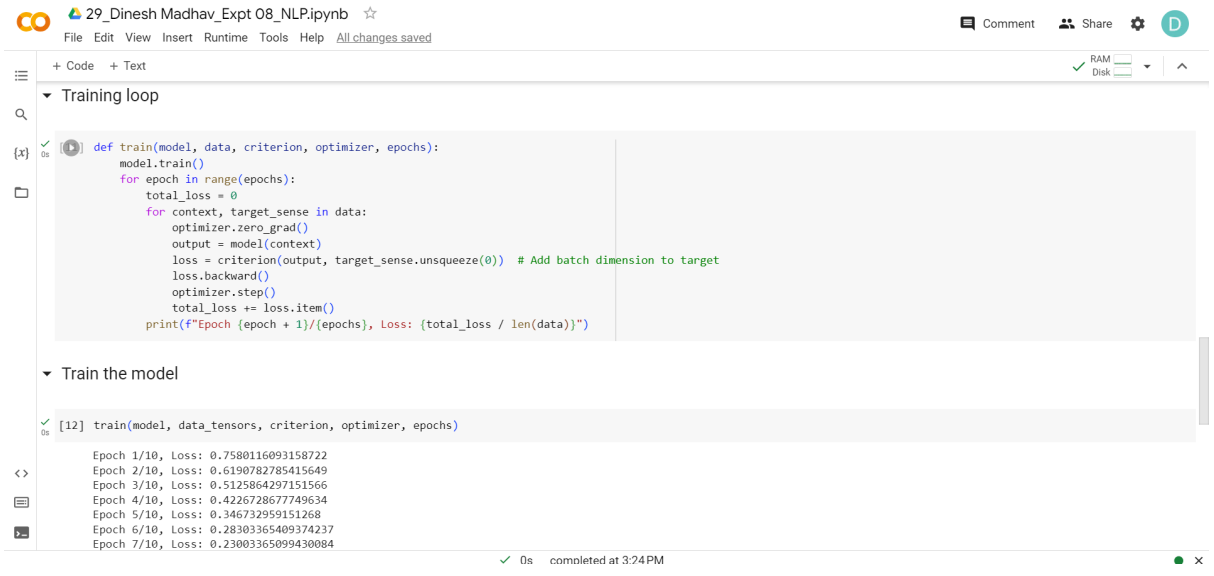
Coreference resolution is an exceptionally versatile tool and can be applied to a variety of NLP tasks such as text understanding, information extraction, machine translation, sentiment analysis, or document summarization. It is a great way to obtain unambiguous sentences which can be much more easily understood by computers.

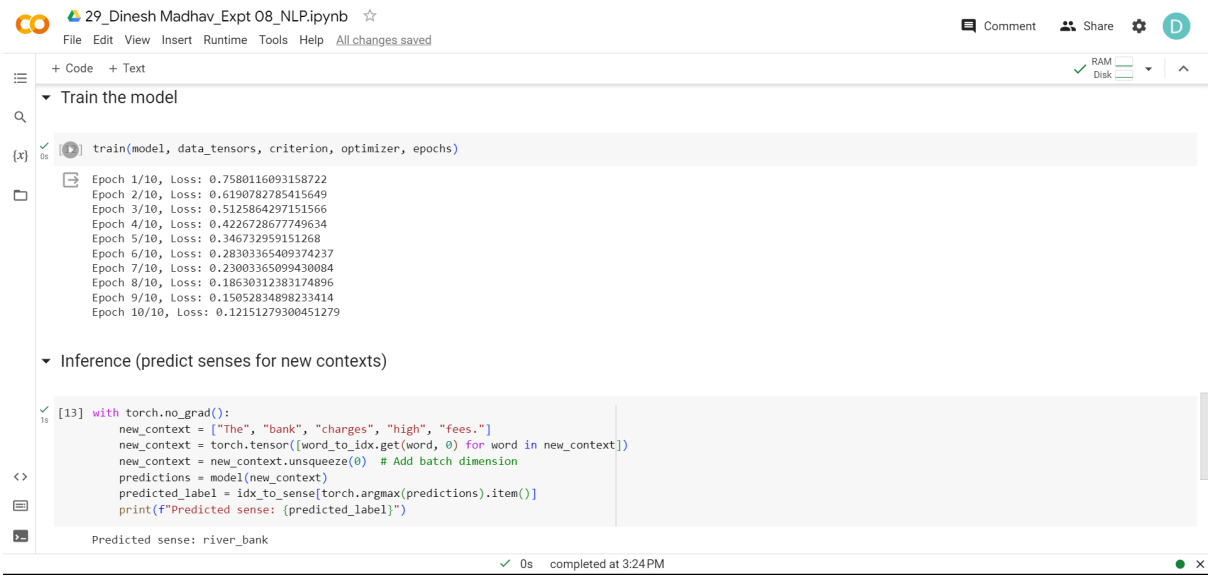
Output:


29_Dinesh Madhav_Expt 08_NLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
29_Dinesh Madhav_Expt08
[2] import torch
import torch.nn as nn
import torch.optim as optim
Sample data (context and senses)
[3] data = [
(["The", "bank", "by", "the", "river", "is", "steep."], "financial_institution"),
(["I", "walked", "along", "the", "river", "bank", "yesterday."], "river_bank"),
]
Create a vocabulary
[4] vocab = set(word for context, _ in data for word in context)
word_to_idx = {word: idx for idx, word in enumerate(vocab)}
idx_to_word = {idx: word for word, idx in word_to_idx.items()}
0s completed at 3:24 PM


29_Dinesh Madhav_Expt 08_NLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Map sense labels to integers
[5] sense_labels = list(set(label for _, label in data))
sense_to_idx = {sense: idx for idx, sense in enumerate(sense_labels)}
idx_to_sense = {idx: sense for sense, idx in sense_to_idx.items()}
Convert data to tensors
[6] data_tensors = [(torch.tensor([word_to_idx[word] for word in context]), torch.tensor(sense_to_idx[sense])) for context, sense in data]
Define the LSTM-based WSD model
[7] class WSDModel(nn.Module):
def __init__(self, vocab_size, embedding_dim, hidden_dim, sense_count):
super(WSDModel, self).__init__()
self.embedding = nn.Embedding(vocab_size, embedding_dim)
self.lstm = nn.LSTM(embedding_dim, hidden_dim)
self.fc = nn.Linear(hidden_dim, sense_count)
def forward(self, context):
embedded = self.embedding(context)
lstm_out, _ = self.lstm(embedded.view(len(context), 1, -1))
prediction = self.fc(lstm_out[-1])


29_Dinesh Madhav_Expt 08_NLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
[7] prediction = self.fc(lstm_out[-1])
return prediction
Hyperparameters
vocab_size = len(vocab)
embedding_dim = 100
hidden_dim = 64
sense_count = len(sense_labels)
learning_rate = 0.001
epochs = 10
Initialize the model
[9] model = WSDModel(vocab_size, embedding_dim, hidden_dim, sense_count)
Define the loss function and optimizer
[10] criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
0s completed at 3:24 PM


29_Dinesh Madhav_Expt 08_NLP.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Training loop
def train(model, data, criterion, optimizer, epochs):
model.train()
for epoch in range(epochs):
total_loss = 0
for context, target_sense in data:
optimizer.zero_grad()
output = model(context)
loss = criterion(output, target_sense.unsqueeze(0)) # Add batch dimension to target
loss.backward()
optimizer.step()
total_loss += loss.item()
print(f"Epoch {epoch + 1}/{epochs}, Loss: {total_loss / len(data)}")
Train the model
[12] train(model, data_tensors, criterion, optimizer, epochs)
Epoch 1/10, Loss: 0.7580116093158722
Epoch 2/10, Loss: 0.6190782785415649
Epoch 3/10, Loss: 0.5125864297151566
Epoch 4/10, Loss: 0.4226728677749634
Epoch 5/10, Loss: 0.346732959151268
Epoch 6/10, Loss: 0.28303365409374237
Epoch 7/10, Loss: 0.23003365099430084
0s completed at 3:24 PM



Conclusion:

The results obtained after resolving coreferences are more accurate than the results without resolving coreferences. This is because resolving coreferences helps to disambiguate the meaning of words and phrases. For example, in the sentence "The bank charges high fees", the word "bank" could refer to either a financial institution or a river bank. However, if we know that the sentence is about banking, then we can resolve the coreferences and determine that the word "bank" refers to a financial institution..