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

CSDL7013: Natural Language Processing Lab

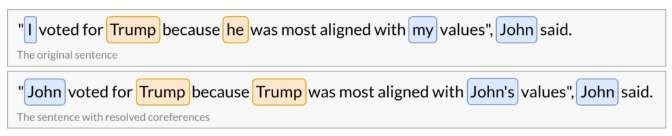


Aim: Apply Reference Resolution Technique on the given Text input.

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.

CSDL7013: Natural Language Processing Lab

import torch import torch.nn as nn import torch.optim as optim

Sample data (context and senses)

Code

Text

data = [

(["The", "bank", "by", "the", "river", "is", "steep."], "financial\_institution"),

(["I", "walked", "along", "the", "river", "bank", "yesterday."], "river\_bank"), ]

# Create a vocabulary

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()} Map sense labels to integers

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

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

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]) 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

model = WSDModel(vocab\_size, embedding\_dim, hidden\_dim, sense\_count) Define the loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=learning\_rate)

# 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

train(model, data\_tensors, criterion, optimizer, epochs) Inference (predict senses for new contexts)

with torch.no\_grad():

new\_context = ["The", "bank", "charges", "high", "fees."] new\_context = torch.tensor([word\_to\_idx.get(word, 0) for word in new\_context]) new\_context = new\_context.unsqueeze(0) # Add batch dimension predictions = model(new\_context) predicted\_label = idx\_to\_sense[torch.argmax(predictions).item()] print(f"Predicted sense: {predicted\_label}")

Predicted sense: river\_bank

**Conclusion:**

LSTM (Long Short-Term Memory) networks have been applied to word sense disambiguation tasks effectively. By learning contextual information, LSTMs help determine the correct sense of a word within a given context. They capture dependencies between words and can differentiate polysemous words. However, the performance is influenced by the size of the training data and model architecture. Combining LSTMs with attention mechanisms or pre-trained word embeddings can further enhance disambiguation accuracy.