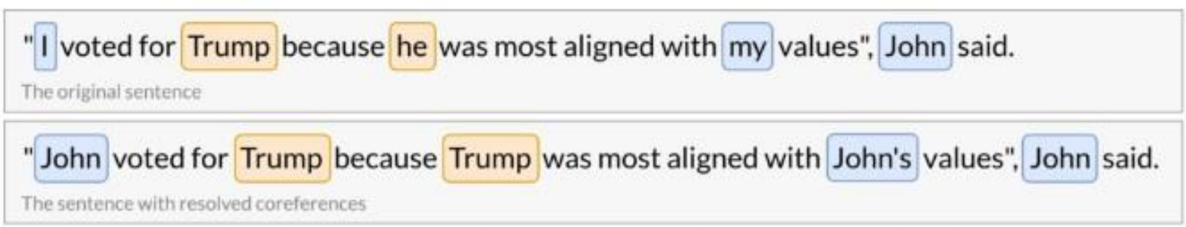
Name: Niyati Patil
Experiment No.8
Implement word sense disambiguation using LSTM/GRU
Date of Performance:
Date of Submission:

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:

In [1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
```

Sample data (context and senses)

In [2]:
```
data = [
    (["The", "bank", "by", "the", "river", "is", "steep."], "financial_institution"),
    (["I", "walked", "along", "the", "river", "bank", "yesterday."], "river_bank"),
]
```

Create a vocabulary

In [3]:
```
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

In [4]:
```
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

In [5]:
```
data_tensors = [(torch.tensor([word_to_idx[word] for word in context]), torch.tensor(sense_to_idx[sense])) for context, sens
```

Define the LSTM-based WSD model

In [6]:
```
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

In [7]:
```
vocab_size = len(vocab)
embedding_dim = 100
hidden_dim = 64
sense_count = len(sense_labels)
learning_rate = 0.001
epochs = 10
```

Initialize the model

In [8]:
```
model = WSDModel(vocab_size, embedding_dim, hidden_dim, sense_count)
```

Define the loss function and optimizer

In [9]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

Training loop

In [10]:
```
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

In [11]:
```
train(model, data_tensors, criterion, optimizer, epochs)
```

```
Epoch 1/10, Loss: 0.6474064588546753
Epoch 2/10, Loss: 0.54472316801548
Epoch 3/10, Loss: 0.46562162041664124
Epoch 4/10, Loss: 0.39604616165161133
Epoch 5/10, Loss: 0.3340972512960434
Epoch 6/10, Loss: 0.2790592089295387
Epoch 7/10, Loss: 0.23062339425086975
Epoch 8/10, Loss: 0.1886119320988655
Epoch 9/10, Loss: 0.15281113982200623
Epoch 10/10, Loss: 0.12287755310535431
```

Inference (predict senses for new contexts)

In [12]:
```
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: financial_institution
```

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.