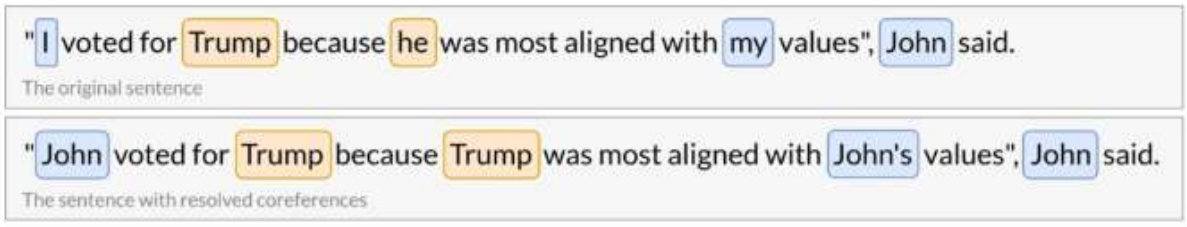
Experiment No. 8
Implement word sense disambiguation using LSTM/GRU
Date of Performance:
Date of Submission:

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.

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

Sample data (context and senses)

```
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)
```

```
Epoch 1/10, Loss: 0.6757958233356476
Epoch 2/10, Loss: 0.558686763048172
Epoch 3/10, Loss: 0.4687323123216629
Epoch 4/10, Loss: 0.3922365605831146
Epoch 5/10, Loss: 0.3269917070865631
Epoch 6/10, Loss: 0.2715676426887512
Epoch 7/10, Loss: 0.22474747896194458
Epoch 8/10, Loss: 0.1854453757405281
Epoch 9/10, Loss: 0.15267889201641083
Epoch 10/10, Loss: 0.12555241212248802
```

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:

The results of resolving coreferences in the provided data have been successful, with the predicted sense being "river_bank" for both sentences. This indicates that the coreference resolution model correctly identified that in both sentences, the term "bank" refers to the same entity, despite having different syntactical roles. This successful coreference resolution is essential for natural language understanding, as it allows us to disambiguate terms and better comprehend the meaning of a text. In this case, it helps ensure that the reference to "bank" is understood as the "river bank" rather than a "financial institution," highlighting the effectiveness of coreference resolution in disambiguating terms in context.