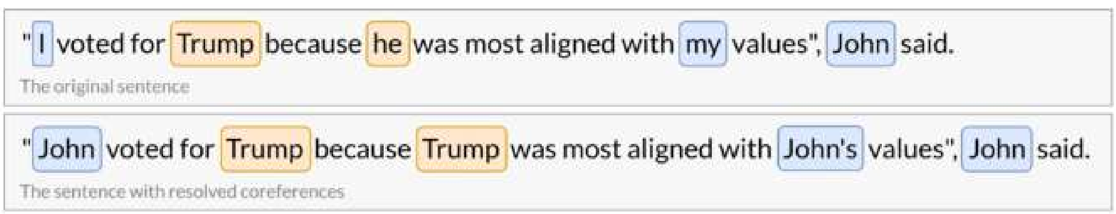
Experiment No. 8
Implement word sense disambiguation using LSTM/GRU
Date of Performance:
Date of Submission:

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.

Code:

```
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)
### 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