EX.NO.: 14
DATE: 14.02.2025

RNN-Based Language Detection and Translation System

To detect the language of the given text and translate the text to english language using

PROCEDURE:

1. Import the necessary libraries
2. Define the texts that require translations
3. Perform the preprocessing on the text
4. Import RR translator and apply the same on the text
5. Display the translated text

CODE AND OUTPUT

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from torch.utils.data import Dataset, DataLoader

# Sample training data for language classification and translation
lang_data = {
    "swahili": ["mwanafunzi wa Taasisi ya Teknolojia ya Coimbatore", "habari za
asubuhi"],
    "english": ["student of Coimbatore Institute of Technology", "good morning"]
}

# Vocabulary and tokenization
def tokenize(text):
    return text.lower().split()

lang_vocab = {word for sentence in lang_data["swahili"] + lang_data["english"] for word
in tokenize(sentence)}
word2idx = {word: idx for idx, word in enumerate(lang_vocab, start=1)}
word2idx["<PAD>"] = 0
idx2word = {idx: word for word, idx in word2idx.items()}
vocab_size = len(word2idx)

# Convert text to numerical representation
def text_to_tensor(text, max_len=10):
    tensor = [word2idx.get(word, 0) for word in tokenize(text)]
    tensor = tensor[:max_len] + [0] * (max_len - len(tensor))
    return torch.tensor(tensor, dtype=torch.long)

# Define Dataset class
class TranslationDataset(Dataset):
    def __init__(self, source_texts, target_texts):
        self.source_tensors = [text_to_tensor(text) for text in source_texts]
        self.target_tensors = [text_to_tensor(text) for text in target_texts]

    def __len__(self):
        return len(self.source_tensors)

    def __getitem__(self, idx):
        return self.source_tensors[idx], self.target_tensors[idx]

# Create dataset and dataloader
dataset = TranslationDataset(lang_data["swahili"], lang_data["english"])
dataloader = DataLoader(dataset, batch_size=2, shuffle=True)

# RNN for Translation
class RNNTranslator(nn.Module):
```

```
    def __init__(self, vocab_size, embed_dim, hidden_dim):
        super(RNNTranslator, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.rnn = nn.RNN(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        output, _ = self.rnn(x)
        return self.fc(output)

translator = RNNTranslator(vocab_size, embed_dim=16, hidden_dim=32)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(translator.parameters(), lr=0.01)

# Training Loop
num_epochs = 100
for epoch in range(num_epochs):
    for src, tgt in dataloader:
        optimizer.zero_grad()
        output = translator(src)
        loss = criterion(output.view(-1, vocab_size), tgt.view(-1))
        loss.backward()
        optimizer.step()

# Translate function
def translate(text):
    with torch.no_grad():
        input_tensor = text_to_tensor(text).unsqueeze(0)
        output_tensor = translator(input_tensor)
        output_indices = output_tensor.argmax(dim=2).squeeze().tolist()
        return " ".join(idx2word.get(idx, "?") for idx in output_indices if idx in
idx2word and idx2word[idx] != "<PAD>")

# Translate a sample text
text = "mwanafunzi wa Taasisi ya Teknolojia ya Coimbatore"
translated_text = translate(text)
print("Translated Text:", translated_text)
```

```
Translated Text: student of coimbatore institute of technology
```