Exp 9: Build a Recurrent Neural Network

Aim: To Build and train a RNN for dataset

objectives:

1) To understand the concept of sequential data and need for RNNs
2) To preprocess texual data into numerical form using embeddings
3) To Implement an RNN model in PyTorch for sentiment classification
4) To evaluate model performance using accuracy & loss

Pseudocode

BEGIN

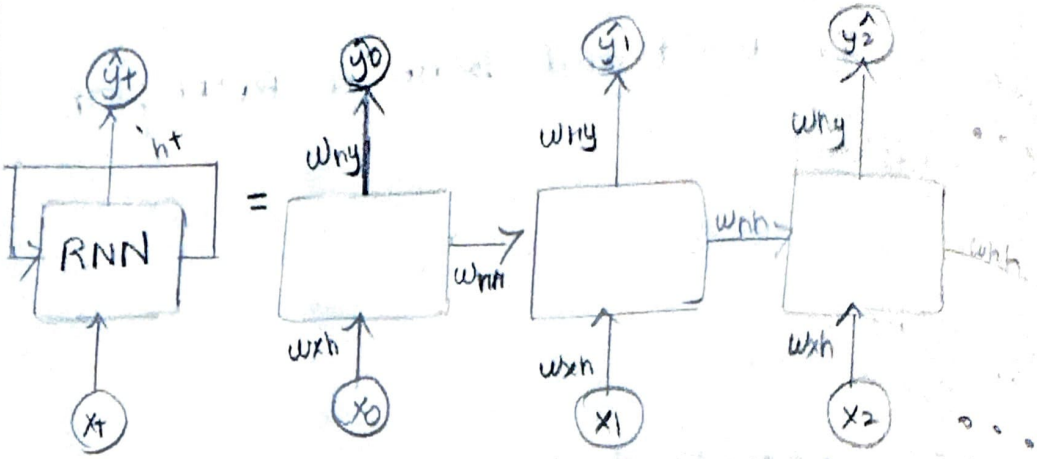
Import required libraries (Torch, Torch.nn)

Load Dataset

IMDB dataset

Preprocess:

- Tokenize sentences
- Convert words to integer indices
- Pad sequence to same length
- Split into train & Test.



$$h_t = f_W(x_t, h_{t-1})$$

cell state — function with weights — input — old state

Output Vector

$$\hat{y}_t = W_{hy}^T h_t$$

Update Vector

$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$

Input Vector

x_t

define RNN model

Create RNN class with:

- Embedding layer
- RNN layer
- Fully connected layer
- Sigmoid activation for binary output

Training

Initialize model, loss function

For each epoch:

For each batch in training data:

Forward Pass

(embedding → RNN → FC → output)

Compute Loss

Back propagate loss

update weights

END FOR

Testing

Evaluate model on Test data

Calculate accuracy

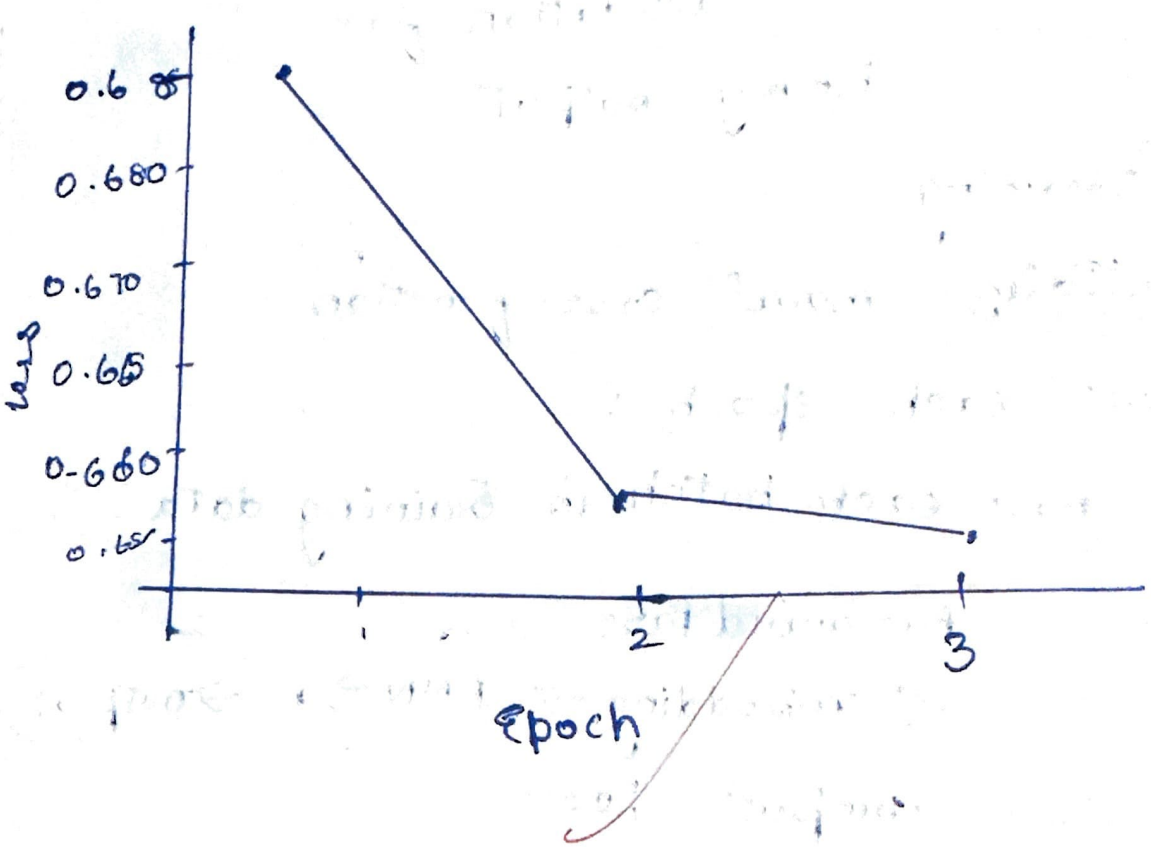
END

Output

Epoch 1, Training Loss : 0.6868

Epoch 2, Training Loss : 0.6962

Epoch 3, Training Loss : 0.6567



Observation

1) The IMDB dataset was preprocessed (tokenized, padded) and fed into RNN for sentiment classification (positive / Negative)

2) The model trained for 8 epochs with training loss decreasing steadily across epochs

3) The RNN was able to capture sequential word patterns, which improved its ability to differentiate between positive and negative reviews.

4) The Final Test accuracy indicates moderate performance and scope for improvement using LSTM or advanced embeddings.

9/10/25

Result

Implemented RNN Recurrent Neural Network.

Code

Notebook Python 3 (ipyl

[1]:
```
import pandas as pd
```

[7]:
```
df=pd.read_csv("IMDB Dataset.csv")
```

[8]:
```
df.head(5)
```

[8]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

[9]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
2025-09-23 08:48:52.757119: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to register cuFFT factory: Attempting to register factory for plugin cuF
hen one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1758597532.923064  381952 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registe
E0000 00:00:1758597532.996783  381952 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been regi
red
W0000 00:00:1758597534.217134  381952 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than or
W0000 00:00:1758597534.217186  381952 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than or
W0000 00:00:1758597534.217194  381952 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than or
W0000 00:00:1758597534.217200  381952 computation_placer.cc:177] computation placer already registered. Please check linkage and avoid linking the same target more than or
2025-09-23 08:48:54.309572: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-criti
operations.
```

[10]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

[11]:
```
df['sentiment'] = df['sentiment'].map({'positive': 1, 'negative': 0})
```

[12]:
```
train_texts, test_texts, train_labels, test_labels = train_test_split(
    df['review'], df['sentiment'], test_size=0.2, random_state=42
)
```

[13]:
```
max_words = 10000
max_len = 200
```

[14]:
```
tokenizer = Tokenizer(num_words=max_words, oov_token="<unk>")
tokenizer.fit_on_texts(train_texts)
```

[15]:
```
X_train = tokenizer.texts_to_sequences(train_texts)
X_test = tokenizer.texts_to_sequences(test_texts)
```

[16]:
```
X_train = pad_sequences(X_train, maxlen=max_len, padding='post')
X_test = pad_sequences(X_test, maxlen=max_len, padding='post')
```

[17]:
```
X_train = torch.tensor(X_train, dtype=torch.long)
y_train = torch.tensor(train_labels.values, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.long)
y_test = torch.tensor(test_labels.values, dtype=torch.long)
```

[18]:
```
train_data = TensorDataset(X_train, y_train)
test_data = TensorDataset(X_test, y_test)
```

[19]:
```
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
test_loader = DataLoader(test_data, batch_size=32)
```

[20]:
```
class RNNClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim):
        super(RNNClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.rnn = nn.RNN(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        embedded = self.embedding(x)
        output, hidden = self.rnn(embedded)
        return self.fc(hidden.squeeze(0))
```

[21]:
```
vocab_size = max_words
embed_dim = 64
hidden_dim = 128
output_dim = 2
```

[22]:
```
model = RNNClassifier(vocab_size, embed_dim, hidden_dim, output_dim)
```

[23]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

[24]:
```
train_losses = []

for epoch in range(3):  # train for 3 epochs
    total_loss = 0
```

[24]:
```
train_losses = []

for epoch in range(3):  # train for 3 epochs
    total_loss = 0
    model.train()
    for texts, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(texts)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_loss = total_loss / len(train_loader)
    train_losses.append(avg_loss)
    print(f"Epoch {epoch+1}, Training Loss: {avg_loss:.4f}")
```

```
Epoch 1, Training Loss: 0.6868
Epoch 2, Training Loss: 0.6562
Epoch 3, Training Loss: 0.6567
```

[25]:
```
model.eval()
correct, total = 0, 0
with torch.no_grad():
    for texts, labels in test_loader:
        outputs = model(texts)
        predictions = torch.argmax(outputs, dim=1)
        correct += (predictions == labels).sum().item()
        total += labels.size(0)

print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

[27]:
```
plt.figure(figsize=(8,5))
plt.plot(range(1, len(train_losses)+1), train_losses, marker='o', color='blue')
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.xticks(range(1, len(train_losses)+1))
plt.grid(True)
plt.show()
```