RA2311047010040

Index

Superior® Note Books

Name : Vishva. S

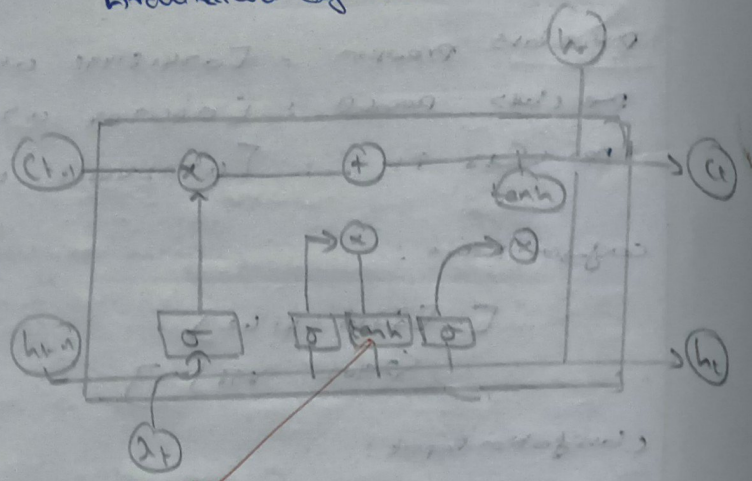
Subject : DL

Std. : ..... Sec. : ..... Roll No. .....

School : .....

No.	Date	Title	Marks	Signature
1	24/7/25	Exploring the DL platforms.		
2	31/7/25	Implement a classifier using open source dataset		
3	7/8/25	Study the classifier with respect to statistical parameters.		
4	14/8/25	Build a simple feed forward neural network to Recognize Handwritten characters		22/8/25
5	22/8/25	Study of Activation functions & Their role.		9/9/25
6	9/9/25	Implement Gradient Descent & Backpropagation in Deep neural network.		16/9/25
7	16/9/25	Build a CNN model to classify cat & dog image.		23/9/25
8	30/9/25	Build a RNN network.		30/9
9	9/10/25	Experiment using LSTM		
10	9/10/25	Autoencoder		
11	9/10/25	Variational Autoencoder		
12	17/10/25	Deep convolutional GAN		

Architecture of LSTM



9/10/25

Ex: 8 Experiment using LSTM

Aim:

To build & train an LSTM (Long-Short Term Memory) model for sequence prediction, demonstrating how LSTM networks handle long-term dependencies in sequential data. (Titanic dataset)

Objectives:

1) To understand the working of LSTM cells & how they overcome vanishing gradient problems.

2) To apply LSTM for time-series or sequence-based prediction tasks (eg. stock prices, text data)

3) To analyze the model's performance in learning temporal relationship.

4) To compare LSTM's effectiveness with traditional RNN models.

Observation

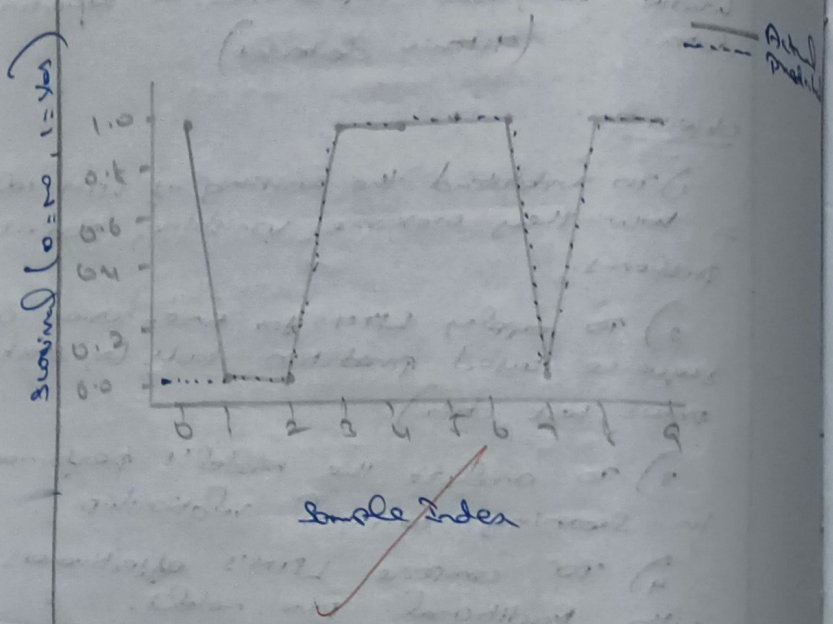
* The LSTM model uses memory cells to store & forget information over long time steps.

* During training, the model learns sequence patterns by adjusting internal gates (inputs, forget, output).

* The accuracy improves as the model captures temporal dependencies in the data.

* The loss decreases gradually, showing effective learning of sequential relationships.

Titanic Survival: Predicted vs Actual



Pseudocode:

BEGIN

Step 1: Import required libraries
(tensorflow/ keras, numpy etc..)

Step 2: load and preprocess sequential dataset
- Normalize data
- Create input sequences (X) and target outputs (Y)
- Split data into training & testing sets.

Step 3: Build LSTM model
- Initialize sequential model
- Add LSTM layer (s) with appropriate number of units.
- Add Dense output layer
- Compile model with optimizer (Adam) and loss function (MSE)

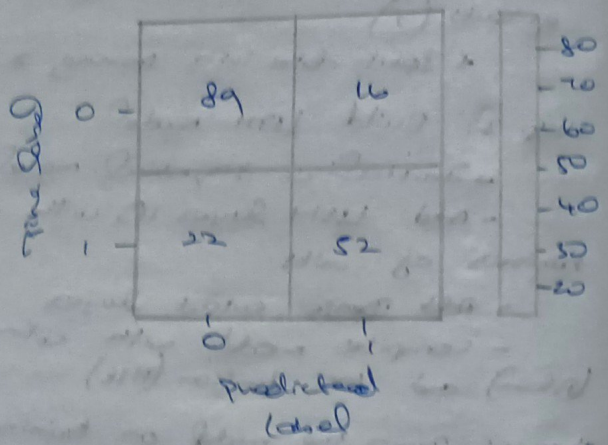
Step 4: Train the model on training data
- Fit model for N epoch
- Monitor training and validation loss

Step 5: Evaluate the model on test data
- Compute prediction accuracy or MSE

Step 6: visualize results
- plot actual vs predicted values

END

Confusion Matrix
Titanic LSTM.



Output:

Epoch [10/50], Loss : 0.626232

Epoch [20/50], Loss : 0.581430

Epoch [30/50], Loss : 0.509261

Epoch [40/50], Loss : 0.473118

Epoch [50/50], Loss : 0.463360.

Predicted vs Actual (first 10 vals)

predicted : 0, Actual : 1

0	0
0	0
1	1
1	1
1	1
1	1
1	0
0	1
1	1
1	

Accuracy on test set : 78.77%.

confusion Matrix :

[[89 16]
[22 52]]

Result:

Successfully Implemented the LSTM on Titanic dataset.

30/7/25

9.) To Build a RNN network.

Aim:

To implement ad train a Recurrent Neural network using pytorch for sentiment classification on the IMDB movie review dataset.

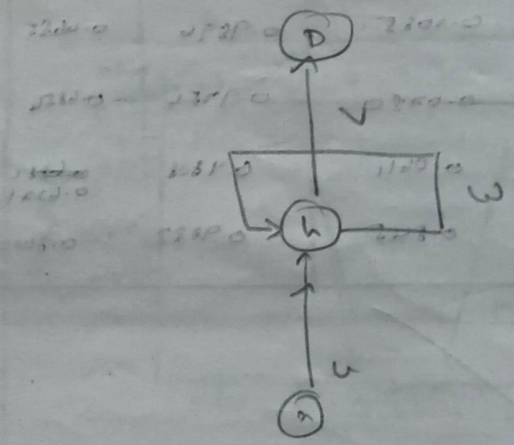
Objectives:

- Load and Preprocess text data using pytorch tools.
- Build a simple RNN model for sequence classification.
- Train the model on the IMDB dataset.
- Evaluate the models accuracy on unseen test data.
- Understand the working of RNN in NLP tasks.

Observation:

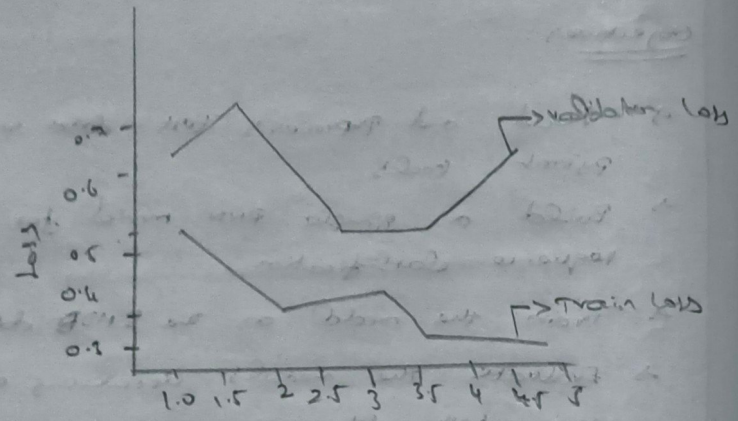
- Tokenization & vocabulary building are crucial preprocessing steps.
- Padding sequences ensures uniform input lengths.
- Using packed sequences improves RNN efficiency when handling variable length inputs.

RNN Architecture



$$h^{(t)} = \sigma\left(W_h(h^{(t-1)}) + W_x e^{(t)} + b\right)$$

Loss vs Epochs



- The model achieves reasonable accuracy on the binary sentiment task.
- Performance may improve with more complex models like LSTM or GRU.

Pseudocode:

```
BEGIN
    Load and tokenize IMDB dataset.
    Build vocabulary from training data
    convert text and labels to numerical form
Pad sequences to equal length in batches.
Define RNN model with embedding, RNN and
    fully connected layers.
Use sigmoid activation for binary classification
Train model for multiple epochs on training
    batches
calculate loss and update weights using
    backpropagation.
Evaluate model accuracy on test data
Output test accuracy
END
```

Output:

Accuracy : 0.7174

Macro precision : 0.7197

Macro Recall : 0.7174

Macro F1 Score = 0.7167

per class precision : [0.74181491 0.69752907]

per class Recall : [0.66704 0.76764]

per class F1 : [0.70246313 0.73097172]

Confusion Matrix :

[[8338 4162]
[2902 9598]]

Classification Report :

	precision	recall	f1-score	support
neg	0.74	0.67	0.70	12500
pos	0.70	0.77	0.73	12500
accuracy			0.72	25000
macro avg	0.72	0.72	0.72	25000
weighted avg	0.72	0.72	0.72	25000

Result:

Successfully implemented the RNN network.

26/9

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Connect T4

RNN

```
# Install datasets if needed
!pip install datasets --quiet

import torch
import torch.nn as nn
import torch.optim as optim
from datasets import load_dataset
from torch.utils.data import DataLoader
from collections import Counter

# 1. Load IMDB dataset with HuggingFace datasets
dataset = load_dataset('imdb')

# 2. Simple tokenizer and vocabulary builder (basic whitespace tokenizer)
def tokenize(text):
    return text.lower().split()

def build_vocab(dataset, min_freq=2):
    counter = Counter()
    for example in dataset:
        counter.update(tokenize(example['text']))
    # Keep tokens with min freq >= min_freq
    vocab = [word for word, freq in counter.items() if freq >= min_freq]
    # Add special tokens
    vocab = ['<pad>', '<unk>'] + sorted(vocab)
    word2idx = {word: idx for idx, word in enumerate(vocab)}
    return word2idx

# Build vocab from train split
word2idx = build_vocab(dataset['train'])

# 3. Encode text into list of token ids
def encode(text):
    tokens = tokenize(text)
    return [word2idx.get(token, word2idx['<unk>']) for token in tokens]

# 4. Dataset class for PyTorch
class IMDBDataset(torch.utils.data.Dataset):
    def __init__(self, split):
        self.data = dataset[split]
```

Variables Terminal

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Connect

```
    return total_loss / total_samples, total_correct / total_samples

# 10. Evaluation loop
def evaluate():
    model.eval()
    total_loss = 0
    total_correct = 0
    total_samples = 0

    with torch.no_grad():
        for texts, labels, lengths in test_loader:
            texts, labels, lengths = texts.to(device), labels.to(device), lengths.to(device)
            outputs = model(texts, lengths)
            loss = criterion(outputs, labels)
            total_loss += loss.item() * texts.size(0)
            preds = outputs.argmax(1)
            total_correct += (preds == labels).sum().item()
            total_samples += texts.size(0)

    return total_loss / total_samples, total_correct / total_samples

# 11. Run training for a few epochs
num_epochs = 10
for epoch in range(num_epochs):
    train_loss, train_acc = train_epoch()
    val_loss, val_acc = evaluate()
    print(f"Epoch {epoch+1} | Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")
```

```
Epoch 1 | Train Loss: 0.6895 | Train Acc: 0.5416 | Val Loss: 0.7014 | Val Acc: 0.5135
Epoch 2 | Train Loss: 0.6788 | Train Acc: 0.5614 | Val Loss: 0.6759 | Val Acc: 0.5624
Epoch 3 | Train Loss: 0.6482 | Train Acc: 0.6084 | Val Loss: 0.6685 | Val Acc: 0.5797
Epoch 4 | Train Loss: 0.6111 | Train Acc: 0.6494 | Val Loss: 0.6619 | Val Acc: 0.6020
Epoch 5 | Train Loss: 0.5648 | Train Acc: 0.6990 | Val Loss: 0.6381 | Val Acc: 0.6383
Epoch 6 | Train Loss: 0.5097 | Train Acc: 0.7450 | Val Loss: 0.6577 | Val Acc: 0.6595
Epoch 7 | Train Loss: 0.4540 | Train Acc: 0.7866 | Val Loss: 0.6276 | Val Acc: 0.6827
Epoch 8 | Train Loss: 0.3974 | Train Acc: 0.8252 | Val Loss: 0.6354 | Val Acc: 0.6966
Epoch 9 | Train Loss: 0.3680 | Train Acc: 0.8410 | Val Loss: 0.6591 | Val Acc: 0.5738
Epoch 10 | Train Loss: 0.4040 | Train Acc: 0.8181 | Val Loss: 0.7398 | Val Acc: 0.6899
```

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, confusion_matrix, classification_report
```

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Connect T4

```
        def forward(self, text, lengths):
            embedded = self.embedding(text)
            packed_emb = nn.utils.rnn.pack_padded_sequence(embedded, lengths.cpu(), batch_first=True, enforce_sorted=False)
            packed_out, hidden = self.rnn(packed_emb)
            out = self.fc(hidden.squeeze(0))
            return out

    # 8. Setup model and training
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    vocab_size = len(word2idx)
    embed_dim = 64
    hidden_dim = 128
    output_dim = 2
    pad_idx = word2idx['<pad>']

    model = RNNClassifier(vocab_size, embed_dim, hidden_dim, output_dim, pad_idx).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-3)

    # 9. Training loop
    def train_epoch():
        model.train()
        total_loss = 0
        total_correct = 0
        total_samples = 0

        for texts, labels, lengths in train_loader:
            texts, labels, lengths = texts.to(device), labels.to(device), lengths.to(device)
            optimizer.zero_grad()
            outputs = model(texts, lengths)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            total_loss += loss.item() * texts.size(0)
            preds = outputs.argmax(1)
            total_correct += (preds == labels).sum().item()
            total_samples += texts.size(0)

        return total_loss / total_samples, total_correct / total_samples

    # 10. Evaluation loop
    def evaluate():
        model.eval()
```

Commands | + Code | + Text | Run all | Connect

```
word2idx = build_vocab(dataset['train'])

# 3. Encode text into list of token ids
def encode(text):
    tokens = tokenize(text)
    return [word2idx.get(token, word2idx['<unk>']) for token in tokens]

# 4. Dataset class for Pytorch
class IMDBDataset(torch.utils.data.Dataset):
    def __init__(self, split):
        self.data = dataset[split]

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

    def __getitem__(self, idx):
        text_ids = encode(self.data[idx]['text'])
        label = self.data[idx]['label']
        return torch.tensor(text_ids, dtype=torch.long), label

# 5. Collate fn for padding batches
def collate_batch(batch):
    texts, labels = zip(*batch)
    lengths = torch.tensor([len(t) for t in texts])
    texts_padded = nn.utils.rnn.pad_sequence(texts, batch_first=True, padding_value=word2idx['<pad>'])
    labels = torch.tensor(labels)
    return texts_padded, labels, lengths

# 6. Create dataloaders
batch_size = 64
train_dataset = IMDBDataset('train')
test_dataset = IMDBDataset('test')

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, collate_fn=collate_batch)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, collate_fn=collate_batch)

# 7. Define RNN Model
class RNNClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim, pad_idx):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=pad_idx)
        self.rnn = nn.RNN(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, text, lengths):
```

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

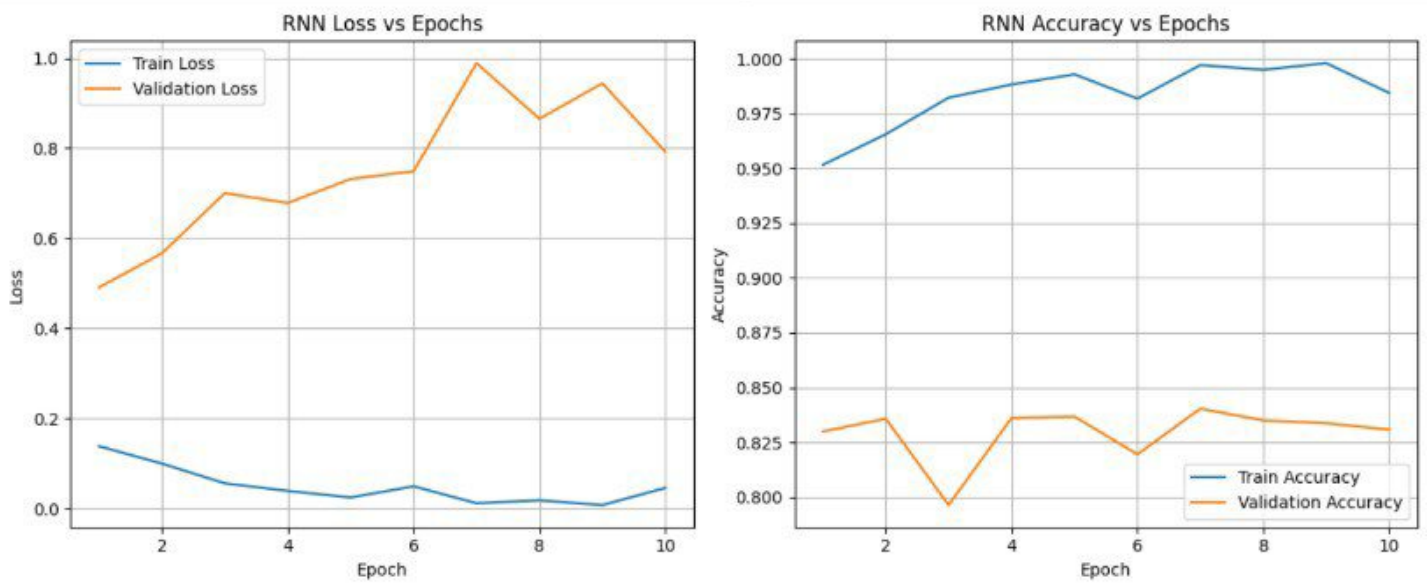
Commands + Code + Text Run all

Connect T4

```
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs+1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, num_epochs+1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('RNN Accuracy vs Epochs')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



LSTM

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all Connect T4

```
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=['neg', 'pos']))
```

```
Accuracy: 0.6899
Macro Precision: 0.6975
Macro Recall: 0.6899
Macro F1 Score: 0.6869

Per-class Precision: [0.73623967 0.65873069]
Per-class Recall: [0.59176 0.788  ]
Per-class F1: [0.65614051 0.71759006]

Confusion Matrix:
[[7397 5103]
 [2650 9850]]

Classification Report:
              precision    recall  f1-score   support

         neg       0.74      0.59      0.66     12500
         pos       0.66      0.79      0.72     12500

    accuracy                           0.69     25000
   macro avg       0.70      0.69      0.69     25000
weighted avg       0.70      0.69      0.69     25000
```

```
!pip install matplotlib --quiet
```

```
import matplotlib.pyplot as plt


plt.figure(figsize=(12,5))

plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs+1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs+1), val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('RNN Loss vs Epochs')
plt.legend()
plt.grid(True)
```

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Connect

```
def get_all_preds_and_labels(model, dataloader):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for texts, labels, lengths in dataloader:
            texts, labels, lengths = texts.to(device), labels.to(device), lengths.to(device)
            outputs = model(texts, lengths)
            preds = outputs.argmax(1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_preds), np.array(all_labels)

# Get preds and labels on test set
y_pred, y_true = get_all_preds_and_labels(model, test_loader)

# Accuracy
acc = accuracy_score(y_true, y_pred)

# Precision, Recall, F1 (macro and per-class)
precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred, average='macro')
precision_per_class, recall_per_class, f1_per_class, _ = precision_recall_fscore_support(y_true, y_pred, average=None)

# Confusion Matrix
conf_mat = confusion_matrix(y_true, y_pred)

print(f"Accuracy: {acc:.4f}")
print(f"Macro Precision: {precision:.4f}")
print(f"Macro Recall: {recall:.4f}")
print(f"Macro F1 Score: {f1:.4f}\n")

print("Per-class Precision:", precision_per_class)
print("Per-class Recall:", recall_per_class)
print("Per-class F1:", f1_per_class)

print("\nConfusion Matrix:")
print(conf_mat)

print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=['neg', 'pos']))
```

```
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            total_loss += loss.item() * texts.size(0)
            preds = outputs.argmax(1)
            total_correct += (preds == labels).sum().item()
            total_samples += texts.size(0)

    return total_loss / total_samples, total_correct / total_samples
```

```
# 10. Evaluation loop for LSTM
def evaluate_lstm(model, dataloader, criterion):
    model.eval()
    total_loss = 0
    total_correct = 0
    total_samples = 0

    with torch.no_grad():
        for texts, labels, lengths in dataloader:
            texts, labels, lengths = texts.to(device), labels.to(device), lengths.to(device)
            outputs = model(texts, lengths)
            loss = criterion(outputs, labels)
            total_loss += loss.item() * texts.size(0)
            preds = outputs.argmax(1)
            total_correct += (preds == labels).sum().item()
            total_samples += texts.size(0)

    return total_loss / total_samples, total_correct / total_samples
```

```
from torch.utils.data import DataLoader
import torch.nn as nn

# 2. Simple tokenizer and vocabulary builder (basic whitespace tokenizer)
def tokenize(text):
    return text.lower().split()

# 3. Encode text into list of token ids
def encode(text):
    tokens = tokenize(text)
    # word2idx is available from previous cells
    return [word2idx.get(token, word2idx['<unk>']) for token in tokens]
```

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all Connect T4

```
hidden_dim = 128
output_dim = 2

# 8. Setup LSTM model and training
lstm_model = LSTMClassifier(vocab_size, embed_dim, hidden_dim, output_dim, pad_idx).to(device)
lstm_criterion = nn.CrossEntropyLoss()
lstm_optimizer = optim.Adam(lstm_model.parameters(), lr=1e-3)

print("LSTM model initialized and moved to device.")
print("Loss function and optimizer defined.")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 7.81k/? [00:00<00:00, 697kB/s]
plain_text/train-00000-of-00001.parquet: 100% 21.0M/21.0M [00:00<00:00, 29.8MB/s]
plain_text/test-00000-of-00001.parquet: 100% 20.5M/20.5M [00:00<00:00, 139kB/s]
plain_text/unsupervised-00000-of-00001.p(…): 100% 42.0M/42.0M [00:00<00:00, 27.2MB/s]
Generating train split: 100% 25000/25000 [00:00<00:00, 89125.45 examples/s]
Generating test split: 100% 25000/25000 [00:00<00:00, 131660.10 examples/s]
Generating unsupervised split: 100% 50000/50000 [00:00<00:00, 160522.58 examples/s]
LSTM model initialized and moved to device.
Loss function and optimizer defined.
```

```
# 9. Training loop for LSTM
def train_lstm_epoch(model, dataloader, criterion, optimizer):
    model.train()
    total_loss = 0
    total_correct = 0
    total_samples = 0

    for texts, labels, lengths in dataloader:
        texts, labels, lengths = texts.to(device), labels.to(device), lengths.to(device)
        optimizer.zero_grad()
        outputs = model(texts, lengths)
        loss = criterion(outputs, labels)
        loss.backward()
```

Variables Terminal

LSTM

```
import torch.nn as nn

# 7. Define LSTM Model
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim, pad_idx):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=pad_idx)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True) # Use nn.LSTM
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, text, lengths):
        embedded = self.embedding(text)
        packed_emb = nn.utils.rnn.pack_padded_sequence(embedded, lengths.cpu(), batch_first=True, enforce_sorted=False)
        packed_out, (hidden, cell) = self.lstm(packed_emb)
        out = self.fc(hidden.squeeze(0))
        return out
```

```
from datasets import load_dataset

# 1. Load IMDB dataset with HuggingFace datasets
dataset = load_dataset('imdb')

# 2. Simple tokenizer and vocabulary builder (basic whitespace tokenizer)
def tokenize(text):
    return text.lower().split()

def build_vocab(dataset, min_freq=2):
    counter = Counter()
    for example in dataset:
        counter.update(tokenize(example['text']))
    # Keep tokens with min freq >= min_freq
    vocab = {word for word, freq in counter.items() if freq >= min_freq}
    # Add special tokens
    vocab = ['<pad>', '<unk>'] + sorted(vocab)
    word2idx = {word: idx for idx, word in enumerate(vocab)}
    return word2idx

# Build vocab from train split
word2idx = build_vocab(dataset['train'])
```

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

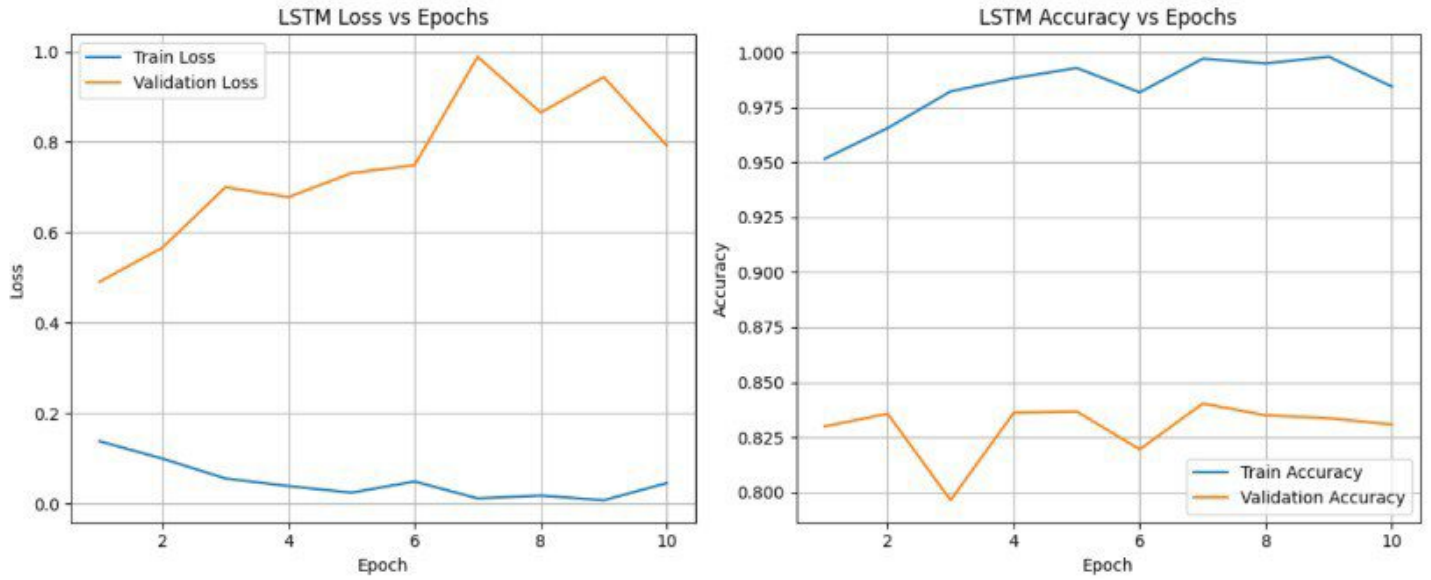
Share

Commands + Code + Text Run all

Connect T4

```
plt.subplot(1, 2, 2)
plt.plot(range(1, num_lstm_epochs+1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, num_lstm_epochs+1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('LSTM Accuracy vs Epochs')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect T4

```
def get_all_preds_and_labels(model, dataloader):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for texts, labels, lengths in dataloader:
            # Ensure tensors are on the correct device
            texts, labels, lengths = texts.to(device), labels.to(device), lengths.to(device)
            outputs = model(texts, lengths)
            preds = outputs.argmax(1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_preds), np.array(all_labels)


# Ensure the model is on the correct device
lstm_model.to(device)

# Get preds and labels on test set
y_pred_lstm, y_true_lstm = get_all_preds_and_labels(lstm_model, test_loader)


print("Classification Report for LSTM Model:")
print(classification_report(y_true_lstm, y_pred_lstm, target_names=['neg', 'pos']))
```

```
Classification Report for LSTM Model:
              precision    recall  f1-score   support

         neg       0.85      0.78      0.82     12500
         pos       0.80      0.87      0.83     12500

    accuracy                           0.82     25000
   macro avg       0.83      0.82      0.82     25000
weighted avg       0.83      0.82      0.82     25000
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))

plt.subplot(1, 2, 1)
plt.plot(range(1, num_lstm_epochs+1), train_losses, label='Train Loss')
```

lab_8&9_RNN.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all Connect T4

```
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, collate_fn=collate_batch)

# Initialize lists to store metrics
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# Define the number of training epochs
num_lstm_epochs = 10

# Loop through the specified number of epochs
for epoch in range(num_lstm_epochs):
    # Train for one epoch
    train_loss, train_acc = train_lstm_epoch(lstm_model, train_loader, lstm_criterion, lstm_optimizer)
    # Append training metrics
    train_losses.append(train_loss)
    train_accuracies.append(train_acc)

    # Evaluate on validation set
    val_loss, val_acc = evaluate_lstm(lstm_model, test_loader, lstm_criterion)
    # Append validation metrics
    val_losses.append(val_loss)
    val_accuracies.append(val_acc)

    # Print epoch progress
    print(f"Epoch {epoch+1} | Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.4f}")
```

```
Epoch 1 | Train Loss: 0.1378 | Train Acc: 0.9516 | Val Loss: 0.4905 | Val Acc: 0.8299
Epoch 2 | Train Loss: 0.0993 | Train Acc: 0.9656 | Val Loss: 0.5663 | Val Acc: 0.8357
Epoch 3 | Train Loss: 0.0554 | Train Acc: 0.9823 | Val Loss: 0.7001 | Val Acc: 0.7963
Epoch 4 | Train Loss: 0.0386 | Train Acc: 0.9883 | Val Loss: 0.6781 | Val Acc: 0.8361
Epoch 5 | Train Loss: 0.0240 | Train Acc: 0.9930 | Val Loss: 0.7314 | Val Acc: 0.8367
Epoch 6 | Train Loss: 0.0488 | Train Acc: 0.9819 | Val Loss: 0.7488 | Val Acc: 0.8195
Epoch 7 | Train Loss: 0.0112 | Train Acc: 0.9972 | Val Loss: 0.9888 | Val Acc: 0.8403
Epoch 8 | Train Loss: 0.0176 | Train Acc: 0.9950 | Val Loss: 0.8653 | Val Acc: 0.8350
Epoch 9 | Train Loss: 0.0072 | Train Acc: 0.9981 | Val Loss: 0.9439 | Val Acc: 0.8337
Epoch 10 | Train Loss: 0.0452 | Train Acc: 0.9845 | Val Loss: 0.7922 | Val Acc: 0.8308
```

```
from sklearn.metrics import classification_report
import numpy as np
import torch

def get_all_preds_and_labels(model, dataloader):
```

Variables Terminal