

# NLP LAB ASSIGNMENT

## Learning Objectives

After completing this lab, students will be able to:

1. Understand limitations of traditional RNNs
2. Implement **LSTM and GRU** architectures
3. Apply LSTM/GRU to **multiple NLP tasks**
4. Compare performance across tasks
5. Analyze results using proper NLP metrics

## NLP Problems Covered (Core Curriculum)

No Problem	Model
1 Text Classification	LSTM / GRU
2 Sentiment Analysis	LSTM / GRU
3 Sequence Labeling (POS / NER)	BiLSTM
4 Language Modeling	LSTM
5 Text Generation	LSTM
6 Question Classification	GRU

## Dataset Suggestions (Standard)

- IMDb / SST-2 → Sentiment
- AG News → Text classification
- CoNLL-2003 → NER
- Penn Treebank → Language modeling

## 1 Common Utilities (Tokenization, Dataset)

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

# Simple tokenizer
def tokenize(text):
    return text.lower().split()

class TextDataset(Dataset):
    def __init__(self, texts, labels, vocab):
        self.data = []
        for t, l in zip(texts, labels):
            ids = [vocab.get(w, vocab["<unk>"]) for w in tokenize(t)]
            self.data.append((ids, l))

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

    def __getitem__(self, idx):
        return self.data[idx]
```

## 2 LSTM / GRU Model (Reusable)

```
class RNNModel(nn.Module):

    def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim, rnn_type="lstm"):

        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embed_dim)

        if rnn_type == "lstm":

            self.rnn = nn.LSTM(embed_dim, hidden_dim, batch_first=True)

        else:

            self.rnn = nn.GRU(embed_dim, hidden_dim, batch_first=True)

        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):

        emb = self.embedding(x)

        h = self.rnn(emb)

        if isinstance(h, tuple): # LSTM

            h = h[0]

        return self.fc(h[-1])
```

### 3 Problem 1: Text Classification

```
texts = [  
    "deep learning is powerful",  
    "nlp is fun",  
    "this movie was terrible",  
    "excellent acting and story"  
]  
  
labels = [1, 1, 0, 1]  
  
vocab = {"<pad>":0, "<unk>":1}  
  
for t in texts:  
    for w in tokenize(t):  
        vocab.setdefault(w, len(vocab))  
  
dataset = TextDataset(texts, labels, vocab)  
loader = DataLoader(dataset, batch_size=2, shuffle=True)  
  
model = RNNModel(len(vocab), 64, 64, 2, rnn_type="lstm")  
loss_fn = nn.CrossEntropyLoss()  
optimizer = optim.Adam(model.parameters(), lr=0.001)  
  
for epoch in range(10):  
    for x, y in loader:  
        x = torch.nn.utils.rnn.pad_sequence(x, batch_first=True)  
        preds = model(x)  
        loss = loss_fn(preds, y)  
        optimizer.zero_grad()  
        loss.backward()
```

```
optimizer.step()  
  
print("Text classification training complete")
```

## Problem 2: Sentiment Analysis (GRU)

Just switch model type:

```
model = RNNModel(len(vocab), 64, 64, 2, rnn_type="gru")
```

- ✓ Same pipeline
- ✓ Shows architectural comparison

## 5 Problem 3: Sequence Labeling (BiLSTM – POS / NER)

```
class BiLSTMTagger(nn.Module):

    def __init__(self, vocab_size, tagset_size):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 64)
        self.lstm = nn.LSTM(64, 64, bidirectional=True, batch_first=True)
        self.fc = nn.Linear(128, tagset_size)

    def forward(self, x):
        emb = self.embedding(x)
        out, _ = self.lstm(emb)
        return self.fc(out)
```

Explain:

- Token-level prediction
- Softmax per timestep
- Used in POS / NER

#### 6 Problem 4: Language Modeling

```
class LanguageModel(nn.Module):  
  
    def __init__(self, vocab_size):  
        super().__init__()  
        self.embed = nn.Embedding(vocab_size, 64)  
        self.lstm = nn.LSTM(64, 128, batch_first=True)  
        self.fc = nn.Linear(128, vocab_size)  
  
  
    def forward(self, x):  
        x = self.embed(x)  
        out, _ = self.lstm(x)  
        return self.fc(out)
```

## 7 Problem 5: Text Generation

```
def generate(model, start_token, vocab, inv_vocab, steps=10):  
    model.eval()  
  
    inp = torch.tensor([[vocab[start_token]]])  
    result = [start_token]  
  
  
    for _ in range(steps):  
        out = model(inp)  
  
        next_id = torch.argmax(out[0, -1]).item()  
        result.append(inv_vocab[next_id])  
  
        inp = torch.tensor([[next_id]])  
  
  
    return " ".join(result)
```

## **8 Problem 6: Question Classification (GRU)**

Classes:

- What
- Why
- How
- When

Same classifier pipeline → different labels.

 **Evaluation Metrics**

<b>Task</b>	<b>Metric</b>
Classification	Accuracy, F1
Sequence labeling	Token F1
LM	Perplexity
Generation	Human evaluation