Machine Learning
LABORATORY: LSTM-RNN In Class

NAME:	STUDENT ID#:

Objectives:

- Students will implement a Recurrent Neural Network (RNN) from scratch, without using built-in RNN APIs.
- Students will manually implement:
  - Hidden state updates over a sequence, a simple linear output layer for sentence classification and manual Stochastic Gradient Descent (SGD) updates.
- Students will understand how RNN hidden states summarize past information.
- Students will visualize the loss curve during training and observe model predictions for simple sentences.

Part 1. Instruction

In this assignment, you will implement a basic Vanilla Recurrent Neural Network (RNN) for simple sequence classification using PyTorch, but without using any high-level RNN modules (no *nn.RNN,* no *optim.SGD* , etc.).

You will manually implement:
- A step-by-step hidden state update based on the RNN equation.
- A manual simple output layer to predict whether a sentence is "good" or "bad".
- Manual parameter updates using basic gradient descent.

The general RNN computation is as follows:

$$s_t = \varphi\left(Ws_{t-1} + Ux_t + b\right)$$

After the final time step, you apply an output layer:

$$logits = W_{out}s_T + b_{out}$$

where:

- $W_{out} \in R^{2 \times n}$ = output weight matrix
- $b_{out} \in R^{2 \times 1}$ = output bias
- The prediction is obtained by applying argmax over the logits.

Part 2. Code Template

Step	Procedure
1	import torch import matplotlib.pyplot as plt # 1. Vocabulary vocab = { "The": 0, "movie": 1, "is": 2, "good": 3, "bad": 4} def word_to_onehot(word): vec = torch.zeros(len(vocab), 1)

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

	``python
	    vec[vocab[word]] = 1.0
	    return vec
	# 2. Training Samples
	train_data = [
	    (["The", "movie", "is", "good"], 1),
	    (["The", "movie", "is", "bad"], 0)
	]
	# 3. Define RNN Parameters
	n = 2 # hidden size
	m = len(vocab)
	T = 4 # number of words
	
	torch.manual_seed(42)
	``

2	```python
	# Initialize Weights (n x n), (n x m), (n x 1), (2 x n), (2 x 1)
	W = None
	U = None
	b = None
	
	W_out = None
	b_out = None
	
	# 4. Training Setup
	learning_rate = 0.1
	num_epochs = 300
	loss_fn = torch.nn.CrossEntropyLoss()
	
	loss_history = []
	``

3	```python
	# 5. Training Loop
	for epoch in range(num_epochs):
	    total_loss = 0.0
	
	    for sentence, label in train_data:
	        inputs = [word_to_onehot(word) for word in sentence]
	        s_prev = torch.zeros(n, 1)
	
	        for x_t in inputs:
	            # TODO: Compute s_t
	            s_t = None # TODO
	
	            s_prev = s_t
	
	        # Output layer
	        logits = None # TODO
	        logits = logits.view(1, -1)
	
	        # Loss
	        target = torch.tensor([label])
	        loss = loss_fn(logits, target)
	        total_loss += loss.item()
	
	        # Backward
	        loss.backward()
	``

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

<table>
<tr><td></td><td>

```
        # Manual update (SGD)
        with torch.no_grad():
            # TODO: Update W, U, b, W_out, b_out
            pass

            # Zero gradients after updating
            W.grad.zero_()
            U.grad.zero_()
            b.grad.zero_()
            W_out.grad.zero_()
            b_out.grad.zero_()

    loss_history.append(total_loss)

    if (epoch + 1) % 50 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}] - Loss: {total_loss:.4f}")
```

</td></tr>
<tr><td>4</td><td>

```
# 6. Plot Loss
plt.plot(loss_history)
plt.xlabel("Epoch")
plt.ylabel("Total Loss")
plt.title("Training Loss Curve")
plt.grid(True)
plt.show()
```

</td></tr>
<tr><td>5</td><td>

```
# 7. Test After Training
print("\n=== Testing ===")
test_sentences = [
    ["The", "movie", "is", "bad"],
    ["The", "movie", "is", "good"]
]

for test_sentence in test_sentences:
    inputs = [word_to_onehot(word) for word in test_sentence]
    s_prev = torch.zeros(n, 1)

    for x_t in inputs:
        #Forward pass again (same as above)
        s_t = None  # TODO
        s_prev = s_t

    logits = None  # TODO
    prediction = torch.argmax(logits)

    print(f"Sentence: {test_sentence} ➔ Prediction: {prediction.item()}")
```

</td></tr>
</table>

Grading Assignment & Submission (30% Max)

Implementation:
1. (5%) Correctly initialize all weights.
2. (5%) Correctly compute the hidden state update and the output layer (logits).
3. (5%) Correctly update all parameters manually using gradients (manual SGD).
4. (5%) The model correctly predicts "good" = 1 and "bad" = 0 in the test sentences.

Question:

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

5. (5%) What does the hidden state s_t represent at each time step when processing a sequence of words?
6. (5%) Why are RNNs hard to train?

Submission :

1. Report: Provide your screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Labs7 In Class Assignment**). Name your files correctly:
   a. Report: StudentID_Lab7_InClass.pdf
   b. Code: StudentID_Lab7_InClass.py or StudentID_Lab7_InClass.ipynb
4. Deadline: 16:20 PM
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

Results and Discussion:

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan