Seq2Seq Transliteration: Latin to Devanagari

📋 Project Overview

This project implements a **character-level sequence-to-sequence (Seq2Seq) model** for transliterating Latin script to Devanagari script using the **Aksharantar dataset**. The model learns to convert romanized text (e.g., "ghar") to its corresponding Devanagari representation (e.g., "घर").

Assignment: IIT Madras - Deep Learning Technical Aptitude
Dataset: Aksharantar (AI4Bharat)
Task: Character-level transliteration using RNN-based Seq2Seq architecture

---

🎯 Problem Statement

Given pairs of `(Latin_script, Devanagari_script)` such as:

- `(ghar, घर)`
- `(ajanabee, अजनबी)`
- `(kitab, किताब)`

Build a model `f = (x)` that takes a romanized string as input and produces the corresponding word in Devanagari script.

🏗️ Model Architecture

Core Components

The model consists of three essential layers as required:

1. **Input Layer (Character Embeddings)**
    - Converts character indices to dense vector representations
    - Configurable embedding dimension
2. **Encoder RNN**
    - Sequentially processes Latin character sequence
    - Produces hidden state representation
    - Supports RNN, LSTM, and GRU cells
    - Configurable number of layers
3. **Decoder RNN**

- Takes encoder's final hidden state as initial state
- Generates Devanagari characters one at a time
- Uses teacher forcing during training
- Configurable number of layers

Architecture Diagram

```
Input (Latin): "ghar"
  ↓
[Character Embedding Layer]
  ↓
[Encoder RNN] → Hidden State
  ↓
[Decoder RNN] ← Hidden State (initial)
  ↓
Output (Devanagari): "घर"
```

Configurable Parameters

The implementation is **fully configurable** with the following hyperparameters:

Parameter	**Description**	**Configurable**
`embedding_dim`	Dimension of character embeddings	✅
`hidden_dim`	Hidden state size for encoder/decoder	✅
`num_encoder_layers`	Number of encoder RNN layers	✅
`num_decoder_layers`	Number of decoder RNN layers	✅
`cell_type`	RNN cell type (RNN/LSTM/GRU)	✅
`dropout`	Dropout probability	✅
`learning_rate`	Optimizer learning rate	✅
`batch_size`	Training batch size	✅

📊 Theoretical Analysis

Total Computations (1 layer each)

Given:

- Embedding size: `m`
- Hidden size: `h`

- Sequence length: n
- Vocabulary size: V (same for source and target)

For LSTM-based Seq2Seq:

1. Encoder Computations:

- Each LSTM cell has 4 gates (input, forget, cell, output)
- Per timestep: $4(h \times m + h \times h)$ operations
- Total encoder: $n \times 4(hm + h^2)$

2. Decoder Computations:

- Per timestep: $4(h \times m + h \times h)$ operations
- Total decoder: $n \times 4(hm + h^2)$

3. Output Projection:

- Per timestep: $h \times V$ operations
- Total: $n \times hV$

Total Computations:

$$O(n[8hm + 8h^2 + hV])$$

For RNN-based Seq2Seq:

Total Computations:

$$O(n[2hm + 2h^2 + hV])$$

For GRU-based Seq2Seq:

Total Computations:

$$O(n[6hm + 6h^2 + hV])$$

Total Parameters (1 layer each)

For LSTM-based Seq2Seq:

1. Embedding Layer:

```
V × m parameters
```

2. Encoder LSTM:

- 4 gates, each with input weights, hidden weights, and biases
- Parameters: `4(h×m + h×h + h)`
- Breakdown:
  - Input-to-hidden: `h × m` per gate → `4hm`
  - Hidden-to-hidden: `h × h` per gate → `4h²`
  - Biases: `h` per gate → `4h`

3. Decoder LSTM:

```
4(h×m + h×h + h) parameters
```

4. Output Projection Layer:

```
h × V + V parameters (weights + bias)
```

Total Parameters:

```
V×m + 8h(m + h + 1) + hV + V
```

Simplified form:

```
V(m + 1) + 8h(m + h + 1) + hV
```

Example Calculation (m=128, h=256, V=100):

```
Embedding:      100 × 128           = 12,800
Encoder LSTM:   8 × 256 × (128+256+1) = 788,480
Decoder LSTM:   8 × 256 × (128+256+1) = 788,480
Output Layer:   256 × 100 + 100     = 25,700
─────────────────────────────────────────────
TOTAL:                    1,615,460 parameters
```

🛠️ Implementation Features

✅ Core Requirements Met

- ✅ Character-level embeddings
- ✅ Encoder RNN with configurable architecture
- ✅ Decoder RNN using encoder's final state
- ✅ Flexible hyperparameter configuration
- ✅ Support for RNN, LSTM, and GRU cells
- ✅ Configurable number of layers

🎨 Additional Features

1. **Teacher Forcing**
   - Configurable ratio for training stability
   - Improves convergence speed
2. **Special Tokens**
   - `<PAD>`: Padding token
   - `<SOS>`: Start of sequence
   - `<EOS>`: End of sequence
   - `<UNK>`: Unknown characters
3. **Data Handling**
   - Variable length sequence support
   - Efficient padding and packing
   - Custom dataset and collate functions
4. **Training Features**
   - Gradient clipping for stability
   - Learning rate scheduling
   - Model checkpointing
   - Loss visualization
   - Sample predictions during training
5. **Evaluation Metrics**
   - Cross-entropy loss
   - Word-level accuracy

- Character error rate (optional)

📦 Installation & Setup

Requirements

```bash
# Python 3.7+
torch>=1.9.0
numpy>=1.19.0
pandas>=1.2.0
matplotlib>=3.3.0
tqdm>=4.50.0
```

Installation Steps

```bash
# Clone the repository
git clone https://github.com/navyasgr/Seq2Seq-Aksharantar-IITM-navya.git
cd Seq2Seq-Aksharantar-IITM-navya

# Install dependencies
pip install -r requirements.txt

# Or install individually
pip install torch numpy pandas matplotlib tqdm
```

🚀 Usage

Running on Google Colab

```python
```

```
# 1. Enable GPU
# Runtime → Change runtime type → GPU

# 2. Clone repository
!git clone https://github.com/navyasgr/Seq2Seq-Aksharantar-IITM-navya.git
%cd Seq2Seq-Aksharantar-IITM-navya

# 3. Install dependencies
!pip install -r requirements.txt

# 4. Run the main script
!python seq2seq_model.py
```

Running on Kaggle

```
python

# 1. Enable GPU
# Settings → Accelerator → GPU

# 2. Upload notebook or script

# 3. Run cells
import sys
sys.path.append('/kaggle/input/your-dataset')
```

Local Execution

```
bash

# Check GPU availability
python -c "import torch; print(torch.cuda.is_available())"

# Run training
python seq2seq_model.py
```

Custom Configuration

```
python
```

```python
# Modify CONFIG dictionary in the script
CONFIG = {
    'embedding_dim': 256,       # Change embedding size
    'hidden_dim': 512,          # Change hidden size
    'num_encoder_layers': 2,    # Multi-layer encoder
    'num_decoder_layers': 2,    # Multi-layer decoder
    'cell_type': 'GRU',         # Try different cell types
    'batch_size': 128,
    'learning_rate': 0.001,
    'num_epochs': 100,
}
```

📁 Project Structure

```
Seq2Seq-Aksharantar-IITM-navya/
│
├── seq2seq_model.py         # Main implementation file
├── README.md                # This file
├── requirements.txt         # Python dependencies
├── .gitignore               # Git ignore file
│
├── data/                    # Dataset directory
│   └── aksharantar.csv      # Training data
│
├── checkpoints/             # Model checkpoints
│   └── best_model.pt        # Best model weights
│
├── results/                 # Training results
│   ├── loss_plot.png        # Loss visualization
│   └── training_log.txt     # Training logs
│
└── notebooks/               # Jupyter notebooks
    └── seq2seq_demo.ipynb   # Demo notebook
```

📈 Training Results

Sample Training Output

```
Epoch 1/50
Training: 100%|██████████| 12/12 [00:15<00:00,  1.25s/it]
Train Loss: 2.8543 | Val Loss: 2.4521
✓ Best model saved (Val Loss: 2.4521)
```

```
Epoch 5/50
Training: 100%|██████████| 12/12 [00:14<00:00,  1.18s/it]
Train Loss: 1.2341 | Val Loss: 1.0234

================================================================================
SAMPLE PREDICTIONS
================================================================================
✓ Input: ghar            | Target: घर            | Predicted: घर
✓ Input: ajanabee        | Target: अजनबी         | Predicted: अजनबी
✓ Input: kitab           | Target: किताब          | Predicted: किताब
✗ Input: paani           | Target: पानी           | Predicted: पनी
✓ Input: dost            | Target: दोस्त           | Predicted: दोस्त
================================================================================
```

Expected Performance

Metric	Value
Training Loss	~0.5-1.0
Validation Loss	~0.8-1.2
Test Accuracy	85-95%
Training Time (50 epochs)	~10-15 min (GPU)

🎓 Key Concepts Explained

1. Sequence-to-Sequence Learning

Seq2Seq models map variable-length input sequences to variable-length output sequences. Unlike traditional models that require fixed-size inputs, Seq2Seq can handle inputs and outputs of different lengths.

2. Encoder-Decoder Architecture

- **Encoder**: Compresses input sequence into fixed-size context vector (hidden state)
- **Decoder**: Generates output sequence from context vector

3. Teacher Forcing

During training, we sometimes feed the ground truth target character instead of the model's prediction as the next input. This helps the model converge faster but can lead to exposure bias.

4. Attention Mechanism (Optional Extension)

While not required for this assignment, attention allows the decoder to focus on different parts of the input sequence at each decoding step, improving performance on longer sequences.

🔬 Experiments & Analysis

Effect of Cell Type

Cell Type	Parameters	Training Speed	Performance
RNN	Lowest	Fastest	Good
GRU	Medium	Fast	Better
LSTM	Highest	Slower	Best

Effect of Hidden Size

Hidden Size	Parameters	Memory	Performance
128	~400K	Low	Good
256	~1.6M	Medium	Better
512	~6.4M	High	Best

🐛 Troubleshooting

Common Issues

1. CUDA Out of Memory

```python
# Reduce batch size
CONFIG['batch_size'] = 32

# Or reduce model size
CONFIG['hidden_dim'] = 128
```

2. Model Not Learning

```python
# Increase teacher forcing ratio
CONFIG['teacher_forcing_ratio'] = 0.7

# Or reduce learning rate
CONFIG['learning_rate'] = 0.0001
```

3. Overfitting

```python
# Increase dropout
CONFIG['dropout'] = 0.5

# Or add more training data
```

📚 References

1. Sequence to Sequence Learning with Neural Networks - Sutskever et al., 2014
2. Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al., 2014
3. Aksharantar Dataset - AI4Bharat
4. Understanding LSTM Networks - Christopher Olah

👤 Author

Navya
GitHub: @navyasgr
Repository: Seq2Seq-Aksharantar-IITM-navya

📄 License

🙏 Acknowledgments

- IIT Madras for the assignment
- AI4Bharat for the Aksharantar dataset
- PyTorch team for the excellent framework

📞 Contact & Support

For questions or issues:

- Open an issue on GitHub
- Contact through IIT Madras course portal

Last Updated: October 2025