Assignment 2: The Georgian Spellcheck

Overview

In this assignment, you will build a character-level sequence-to-sequence model that can correct misspelled Georgian words. Your model will learn to transform corrupted text into its correct form, one character at a time.

This is not a contextual spellchecker—your model operates on individual words in isolation, learning the intrinsic patterns of valid Georgian orthography.

The Challenge

Georgian script has a beautiful one-to-one correspondence between sounds and letters, yet typographical errors are inevitable: fingers slip, keys are adjacent, characters get swapped or dropped. Your task is to build a neural network that has internalized what "correct Georgian" looks like at the character level and can repair damage to individual words.

Technical Requirements

1. The Data (The "Foundation")

You must construct your own training dataset. This is a critical part of the assignment.

Data Collection:

- You need a source of correctly spelled Georgian words. The internet is full of Georgian text—news sites, Wikipedia, literature archives, government documents. Be resourceful.
- Your final dataset should contain a substantial vocabulary (think thousands of unique words, not hundreds).

Data Preparation:

- Your model needs to see both correct and incorrect examples to learn the transformation.
- Consider: How do real spelling errors occur? What makes a synthetic error realistic versus artificial?
- Your training pairs should be `(input_word, target_word)` where the target is always the correct spelling.
- Important: Your dataset should include cases where the input is already correct. Why might this matter?

2. The Model (The "Mind")

You must implement a many-to-many recurrent architecture that processes sequences character by character.

Architecture Constraints:

- Use RNN, LSTM, or GRU cells (or a combination)
- The model must be character-level (not subword, not word-level)
- You may use an encoder-decoder structure or a single sequence model—justify your choice
- No pre-trained embeddings or language models

Design Considerations:

- How will you handle the fact that output length may differ from input length?
- What is your character vocabulary? How do you handle the Georgian alphabet?
- Think about special tokens you might need (start, end, padding, unknown)

3. The Pipeline (The "Craft")

Your implementation should demonstrate understanding of the full deep learning workflow:

- Proper train/validation splitting
- Batching and padding strategies for variable-length sequences
- Training loop with loss tracking
- Early stopping or checkpoint saving based on validation performance

Example Behavior

Input: “გამარჰონა” → Output: “გამარჯობა”

Input: "თბილისი" → Output: "თბილისი" (already correct)

Input: "პროგამა" → Output: "პროგრამა"

Deliverables

Notebook 1: `data_and_training.ipynb`

This notebook contains your data pipeline and model training.

Must include:

1. **Data Collection**: Code or clear documentation of how you obtained your word list
2. **Data Generation**: Your methodology for creating training pairs (show examples!)
3. **Model Definition**: Your architecture with brief justification of design choices
4. **Training**: Complete training loop with visible loss curves

5. **Save**: Export your trained model to disk

"Run All" Requirement: The notebook should execute end-to-end, though data collection steps may be cached/pre-downloaded.

Notebook 2: `inference.ipynb`

This notebook demonstrates your model in action.

Must implement:

python

```python
def correct_word(word: str, model_path: str) -> str:
    """
    Takes a potentially misspelled Georgian word and returns the corrected version.
    """
    pass
```

Demonstration: Show your model correcting at least 20 examples (mix of actual errors and correct words).

Model Upload

You must also provide your trained model

Hints and Guidance

- The quality of your synthetic errors matters more than the quantity.
- Georgian keyboard layout might inspire realistic error patterns.
- Think about edit distance—most real typos are small perturbations.
- Character embeddings can be learned; you don't need anything fancy.

What I'm Looking For

I want to see that you understand:

1. How to frame a practical problem as a sequence-to-sequence task
2. The mechanics of recurrent networks at the character level
3. The importance of data quality and thoughtful dataset construction
4. The full lifecycle from raw data to deployed model

Creativity in your approach—especially in data generation—will be rewarded.

Due Date: 14 days from assignment date