HW1: Language Modeling

DUE FRIDAY, 9/19, AT 5 PM

You'll explore and compare three types of character-level language models—n-gram, RNN, and LSTM—using the TinyStories dataset. Each section includes a model implementation task plus a free-response question to deepen understanding.

Setup

1. Clone the starter code from GitHub. The starter code includes lots more implementation-specific details.
2. In addition to starter code for Part 1, Part 2, and Part 3, this repository contains:
   a. `data/`: **train**, **val**idation, and **test** splits in one-sentence-per-line format
   b. `utils.py`: helper functions
3. Submission: Please submit a link to your code *and* responses on Canvas. Your code may be hosted on Github, Kaggle, or another suitable location. Please make sure any necessary sharing permissions are enabled. Any commits/edits for code beyond the deadline will be considered **late**.

Part 1: n-gram model

In this part, you will build two character-level language models: a **unigram model** (n = 1) and a **5-gram model** (n = 5). Each model will be evaluated based on **prediction accuracy**, and you'll answer a reflection question.

You should implement both models from scratch, using only basic Python and `collections` (no external NLP libraries). Smoothing is required to ensure non-zero probabilities.

1. **[4 points]** Implement a character-level **unigram** language model. Apply add-one **smoothing** to handle unseen characters.
2. **[1 point]** Report **accuracy** for both the validation and test sets. For full credit, accuracy must be at least 17% on `val.txt` and `test.txt`.

   Answer: val.txt: 17.03% test.txt: 17.15%
3. **[3 points]** Now extend your model to a **5-gram** language model (i.e., predict the next character based on the previous 4 characters), still using add-one **smoothing**.
4. **[1 point]** Report **accuracy** for both the validation and test sets. For full credit, accuracy must be at least 57% on `val.txt` and `test.txt`.

   Answer: val.txt: 60.35% test.txt: 59.87%

5. **[1 point]** Free response: For each prompt provided in response.txt, report the 100 most likely next characters for the unigram and 5-gram models. Which seems better, and why? What is still lacking?

   Answer: The unigram model outputs mostly meaningless text, since it only reflects the overall frequency of characters and cannot account for context. The 5-gram model is much better: it generates words and phrases that look coherent, because it can condition on the previous 4 characters. However, it still repeats phrases like "the boy who listen" and lacks long-range structure. What is still lacking is the ability to model longer dependencies, avoid repetitive loops, and generate globally coherent text.

Part 2: Vanilla RNN model

In this part, you will build a bare-bones **RNN** model and write a training loop. Much of the code will be reused in Part 3.

1. **[5 points]** Implement a minimal character-level **RNN** with hidden size **128** using PyTorch tensor operations (examples provided in starter code). You may **not** use PyTorch's built-in `torch.nn.RNNCell`, `torch.nn.RNN`, or other fully-implemented functions.
2. **[3 points]** Write **training** and **evaluation** procedures. You should be able to use the same code for Part 3.
   a. For debugging purposes only, try shortening the training data: `read_data('train.txt')[:5]`
3. **[2 points]** Report **accuracy** for `train.txt`. For full credit, accuracy must be at least 58% on `val.txt` and `test.txt`. Include your **saved model** in the submission files.

   Answer: val.txt: 59.51, test.txt: 58.91

Part 3: LSTM model

In this part, you will build a more advanced RNN called an **LSTM**. You may use the same code structure as Part 2, and the training loop and evaluation code should remain the same.

1. **[5 points]** Implement a character-level **LSTM** with hidden size **128** using PyTorch tensor operations (examples provided in starter code). You may **not** use PyTorch's built-in `torch.nn.LSTMCell`, `torch.nn.LSTM`, or similar fully-implemented functions.
2. **[2 points]** Train similarly to the RNN. Report **accuracy** for `train.txt`. For full credit, accuracy must be at least 60% on `val.txt` and `test.txt`.

   Answer: val.txt: 61.90%, test.txt: 60.86%
3. **[3 points]** Free response: For each prompt provided in response.txt, report the 100 most likely next characters for the vanilla RNN and LSTM models. (As

we know, these are not state-of-the-art models. Look closely to find what *does* work well when answering these questions.)

Answer:

RNN: <BOS>"I'm not ready to go," said the stick and said to play with her mom and said to play with her mom and said to play with her mom

<BOS>Lily and Max were best friends. One day she says.<EOS> the stick to play with her mom and said to play with her mom and said to play with her m

<BOS>He picked up the juice andy was so happy to play with her mom and said to play with her mom and said to play with her mom and

<BOS>It was raining, son around and said to play with her mom and said to play with her mom and said to play with her mom a

<BOS>The end of the story waself to play with her mom and said to play with her mom and said

LSTM: <BOS>"I'm not ready to go," said something from the sun was so happy and said, "i want to see the boy named timmy.<EOS> lily.<EOS> lily.<EOS> li

<BOS>Lily and Max were best friends. One day.<EOS> "what is not to the sun was so happy and said, "i want to see the boy named timmy.<EOS> lily.<EOS> lily.<EOS>

<BOS>He picked up the juice andy.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.

<BOS>It was raining, son and the sun was so happy and said, "i want to see the boy named timmy.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.

<BOS>The end of the story was and said, "i want to see the boy named timmy.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lily.<EOS> lil

a. How does coherence compare between the vanilla RNN and LSTM?
   Answer: The vanilla RNN repeats phrases like *"to play with her mom"* and drifts quickly into loops, showing weak long-term memory. The LSTM maintains more variety, sometimes introducing new entities ("the boy named Timmy"), and keeps context longer, but still falls into repetitive loops ("<EOS> lily."). Overall, the LSTM is more coherent and contextually stable than the vanilla RNN.
b. Concretely, how do the neural methods compare with the n-gram models?
   Answer: Both neural models outperform the unigram and 5-gram baselines. Unlike n-grams, they can generate longer words and sentence-like structures without being strictly limited to short contexts. The LSTM especially captures dependencies beyond a fixed window, while the n-gram repeats rigidly ("the boy who listen").

c. What is still lacking? What could help make these models better?
Answer: They still struggle with global coherence, repetition, and maintaining narrative flow. They lack understanding of syntax, semantics, and long-range dependencies. Improvements could include larger hidden sizes, deeper networks, dropout for regularization, and ultimately more advanced architectures, that model longer contexts and reduce repetition.

Congratulations! That's a wrap on HW1. If your models look like they're spiraling towards a quick demise, you're in the right place. Now onto more structure, syntax, and surprises.