Assignment 3

Name :**INNOCENT KISOKA**
Email: **innocent.kisoka@usi.ch**

Deadline: 10 Dec 2023 - 11.59pm

Language models with LSTM

Data (35 points)

1 Question 1

1.1 What is the dataset package? (5 pts)

The dataset package used in this assignment is `datasets`,a library provided by Hugging Face.

1.2 What data type did you have, and how can you work with it? (5 pts)

The dataset I had is similar to a Python dictionary or Pandas DataFrame but optimized for large-scale data operations.

You can work with this data type by:

- Accessing rows using indexing (e.g., `dataset[0]`).
- Slicing subsets of data (e.g., `dataset[:10]` for the first 10 rows).
- Filtering specific rows based on conditions (e.g., filtering only political news).
- Modifying or transforming the data using the `map` function.
- Converting it to other formats like Pandas DataFrame if needed.

1.3 How many columns does the dataset have, and what's in them? (5 pts)

The dataset has six columns with the following content:

- `link`: The URL link to the full news article.
- `headline`: The title or headline of the news article (primary input for the task).

```
Filtered dataset
Dataset({
    features: ['link', 'headline', 'category', 'short_description', 'authors', 'date'],
    num_rows: 35602
})
```

Figure 1: Columns

- `category`: The category of the news article (e.g., politics, sports, entertainment).
- `short_description`: A brief summary or description of the article.
- `authors`: The authors of the article.
- `date`: The publication date of the article.

Question-2

The dataset is filtered to retain only news articles in the `POLITICS` category. I ended up getting 35602 items..

Question-3

Each headline is processed by splitting into lowercase words, creating a list of words for each title.

```
Map: 100%                    35602/35602 [00:04<00:00, 7253.20 examples/s]
First three tokenized headlines:
['biden', 'says', 'u.s.', 'forces', 'would', 'defend', 'taiwan', 'if', 'china', 'invaded', '<EOS>']
['‘beautiful', 'and', 'sad', 'at', 'the', 'same', 'time’:', 'ukrainian', 'cultural', 'festival', 'takes', 'on', 'a', 'deeper', 'meaning', 'this', 'year', '<EOS>']
['biden', 'says', "queen's", 'death', 'left', "'giant", "hole'", 'for', 'royal', 'family', '<EOS>']
```

Figure 2: Tokenized list

Question 4

5 most common words: [('EOS', 35602), ('to', 10701), ('the', 9618), ('trump', 6895) and ('of', 5536)]

Number of unique words I ended up with: 33234

Question 5

Dataset Class

Represents tokenized sequences using `word_to_int`. Each item is a tuple:

– **First part:** Indices for all words except the last one.
– **Second part:** Indices for all words except the first one.

Implementation: Use PyTorch's `Dataset` class, overriding `__len__` and `__getitem__`.

Question 6

Collate Function

Pads shorter sequences with `<PAD>` (ID 0) to match the longest sequence in the batch.

DataLoader

Uses PyTorch's `DataLoader` with the `collate_fn` to batch and pad sequences efficiently for training.

Model Definition (10 pts)

The LSTM-based Model

The model consists of the following components:

– **Embedding Layer:** Converts word indices into dense vector representations.
– **Stacked LSTM:** Captures sequential patterns and long-term dependencies.
– **Dropout Layer:** Prevents overfitting by regularizing between LSTM layers.
– **Fully Connected Layer:** Maps LSTM outputs to the vocabulary size for prediction.
– **Softmax Activation:** Converts logits to probabilities for word prediction.

Initialization Method

`init_state`: Initializes hidden and cell states for LSTM layers based on batch size and number of layers.

Difference Between RNNs and LSTMs

LSTMs include gates (input, forget, output) to manage long-term dependencies, addressing the vanishing gradient problem present in traditional RNNs.

Evaluation - Part 1 (10 pts)

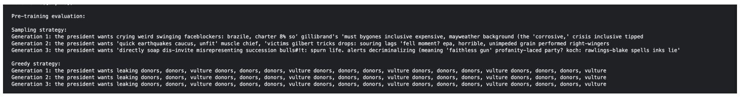
Sentence Completion Strategies

Sentence Generation

Function: `sample(prompt, model, sampling_strategy)`. Generates sentences by iteratively predicting the next word until `<EOS>` is reached.

Example Sentences

Prompt: ”The president wants”.



```
Pre-training evaluation:

Sampling strategy:
Generation 1: the president wants crying weird swinging faceblockers: brazile, charter 8% so' gillibrand's 'must bygones inclusive expensive, mayweather background (the 'corrosive,' crisis inclusive tipped
Generation 2: the president wants 'quick earthquakes caucus, unfit' muscle chief, 'victims gilbert tricks drops: souring lags 'fell moment? epa, horrible, unimpeded grain performed right-wingers
Generation 3: the president wants 'directly soap dis-invite misrepresenting succession bulls#!t: spurn life. alerts decriminalizing (meaning 'faithless gun' profanity-laced party? koch: rawlings-blake spells inks lie'

Greedy strategy:
Generation 1: the president wants leaking donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture
Generation 2: the president wants leaking donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture
Generation 3: the president wants leaking donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture donors, donors, donors, vulture
```

Figure 3: Sentence Generation

Training (35 points)

Standard Training Loop (15 points)

(5 pts) Loss and Perplexity Plot:

A plot of the training loss shows a steady decrease from 4.5277 in epoch 1 to 0.6169 in epoch 12, demonstrating convergence below the target of 1.5. A plot of the perplexity values shows a decrease from 92.5466 in epoch 1 to 1.8531 in epoch 12, further confirming successful training.

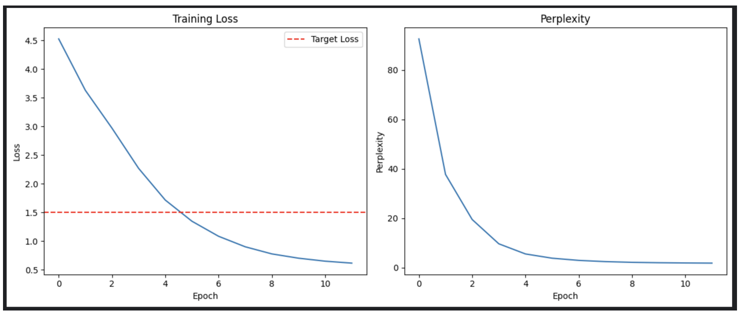
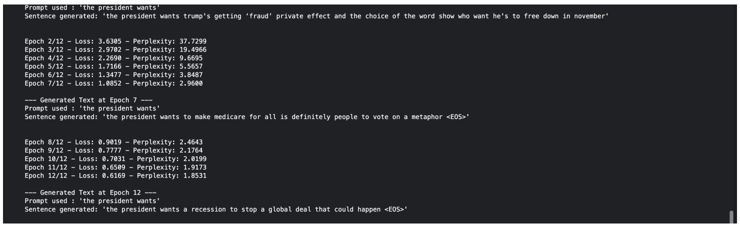


Figure 4: Loss Function and Perplexity Plots



```
Prompt used : 'the president wants'
Sentence generated: 'the president wants trump's getting ‘fraud’ private effect and the choice of the word show who want he's to free down in november'


Epoch 2/12 - Loss: 3.6305 - Perplexity: 37.7299
Epoch 3/12 - Loss: 2.9702 - Perplexity: 19.4966
Epoch 4/12 - Loss: 2.2690 - Perplexity: 9.6695
Epoch 5/12 - Loss: 1.7166 - Perplexity: 5.5657
Epoch 6/12 - Loss: 1.3477 - Perplexity: 3.8487
Epoch 7/12 - Loss: 1.0852 - Perplexity: 2.9600

--- Generated Text at Epoch 7 ---
Prompt used : 'the president wants'
Sentence generated: 'the president wants to make medicare for all is definitely people to vote on a metaphor <EOS>'


Epoch 8/12 - Loss: 0.9019 - Perplexity: 2.4643
Epoch 9/12 - Loss: 0.7777 - Perplexity: 2.1764
Epoch 10/12 - Loss: 0.7031 - Perplexity: 2.0199
Epoch 11/12 - Loss: 0.6509 - Perplexity: 1.9173
Epoch 12/12 - Loss: 0.6169 - Perplexity: 1.8531

--- Generated Text at Epoch 12 ---
Prompt used : 'the president wants'
Sentence generated: 'the president wants a recession to stop a global deal that could happen <EOS>'
```

Figure 5: Generated Text at Epoch 1, 7 12 respectively

(5 pts) Generated Sentences:

After the first epoch: Comment: The generated text is incoherent, indicating the model is still learning basic patterns and relationships.

After the middle epoch (epoch 7): Comment: The model exhibits improved coherence and grammar, demonstrating partial understanding of the prompt context.

At the end of training (epoch 12): Comment: The output is contextually meaningful and well-formed, showcasing the effectiveness of training.

(5 pts) Architecture Justification:

- **Hidden Size (1024):** Large enough to capture complex patterns without overwhelming computational resources.
- **Embedding Dimension (150):** Balances capturing semantic nuances with efficient training.
- **Dropout (0.2):** Prevents overfitting, ensuring better generalization.
- **Gradient Clipping (1.0):** Mitigates exploding gradients, crucial for RNNs.

Truncated Backpropagation Through Time (TBBTT) (20 points)

(10 pts) Observed Differences:

- **Convergence Speed:** TBPTT achieves almost similar loss and perplexity values (1.0431 and 2.8381, respectively) within only 5 epochs, compared to 12 epochs in standard training.
- **Efficiency:** Computationally faster as it processes shorter sequence chunks.
- **Global Dependencies:** May lose some long-term dependencies due to the truncation.

(5 pts) Loss and Perplexity Plot:

- Loss decreases from 4.3043 in epoch 1 to 1.0431 in epoch 5.
- Perplexity reduces from 74.0171 in epoch 1 to 2.8381 in epoch 5.

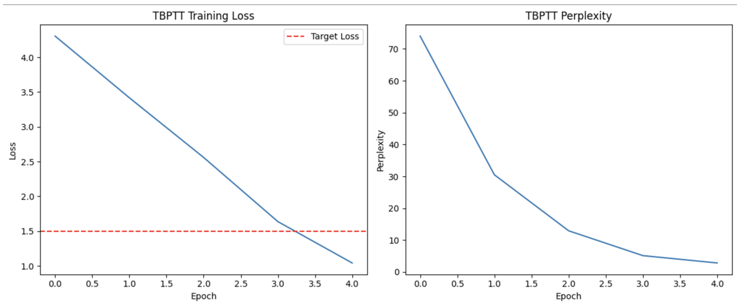


Figure 6: Loss Function and Perplexity Plots after TBPTT

Figure 7: Generated Text at Epoch 1, 3 5 respectively

(5 pts) Generated Sentences:

After the first epoch: Comment: Outputs are basic and lack coherence, indicative of early training.

After the middle epoch (epoch 3): Comment: Outputs improve, showing partial understanding and more logical patterns.

At the end of training (epoch 5): Comment: Outputs are contextually meaningful, showing TBPTT's effectiveness.

Evaluation (5 points)

Sampling Strategy Sentences:

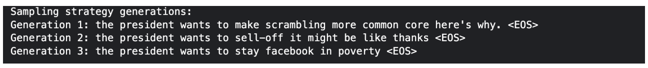


Figure 8: Sampling Strategy

Comment: Sampling produces diverse but inconsistent outputs; not all are meaningful.

Greedy Strategy Sentences:

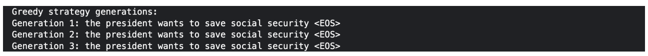


Figure 9: Greedy Strategy

Comment: Greedy decoding generates repetitive but coherent sentences, showing deterministic behavior.

Comparison: Sampling is better for creative outputs, while greedy decoding ensures consistency.

Bonus Question (5 points)

Claim:

The embedding result for `king - man + woman` produces ”woman” instead of the expected ”queen.”

Justification:

The embedding captures some semantic relationships, but the result shows limited analogical reasoning. Possible reasons include:

- Insufficient training data.
- Suboptimal embedding learning.
- Limitations in the architecture’s ability to capture complex word relationships.