Assignment 3

name: EMANUELE BELLINI, email: belliem@usi.ch

Deadline: 5 Dec 2024 - 11.59pm

Language models with LSTM

Data (35 points)

1. Dataset Analysis:

   The Hugging Face datasets `datasets` provides a standard framework for working with machine learning datasets. A number of reasons make this package very useful for our implementation. First, it handles data loading and preprocessing efficiently through its `load_dataset()` function, which returns a structured Dataset object. This object provides a unified interface to access both the training and testing splits of the data.

   Each element in this dataset is a record with multiple fields, represented as Value objects of specific data types (6 columns):

   - `string` type for text fields (link, headline, category, short_description, authors)
   - `timestamp[s]` type for the date field

   Built-in methods simplified the operations on this data structure. The dataset package also provides efficient data streaming capabilities, meaning we could process our 35,602 political articles without loading the entire dataset into memory at once.

2. Vocabulary Analysis:

   - Top 5 most frequent words (excluding special tokens):
     (a) ”to”: 10,701 occurrences
     (b) ”the”: 9,619 occurrences
     (c) ”trump”: 6,896 occurrences
     (d) ”of”: 5,536 occurrences
     (e) ”in”: 5,250 occurrences

     This distribution reflects both common English language structures (with prepositions and articles) and domain-specific content; notably, ”trump” is the third most frequent word. We made the decision to keep all words in our vocabulary for the following reasons:

- Our dataset size (35,602 articles) was manageable enough to maintain a complete vocabulary without significant computational overhead
- Even rare words could have significant semantic meaning in political headlines
- The potential loss of information from removing less frequent words does not justify the minimal benefits of vocabulary reduction

3. Dataset Class Implementation:

   The Dataset class must be implemented to prepare our sequences for training the language model. For each headline sequence, we need to create pairs of input and target sequences that allow the model to learn to predict the next word in a sentence.

   The class takes two inputs: our tokenized sequences and the word_to_int dictionary. For each sequence, it must generate:

   - An input sequence containing all words except the last one, converted to integer indices
   - A target sequence containing all words except the first one, converted to integer indices

   This creates the necessary pairs for the model to learn the relationship between each word and the word that follows it, in our political headlines.

4. Padding, Batches, and DataLoader Implementation:

   The idea of the collate function is to handle variable-length sequences in each batch. It takes a batch of sequences and pad the shorter sequences with the PAD token to the length of the longest sequence of that batch.

   The next step is to set up a PyTorch DataLoader that utilizes the above collate function. The DataLoader internally handles the process of batching and applies our padding function to each batch before feeding it into the model. This ensure that all sequences in one batch have the same length, while avoiding unnecessary padding across the entire dataset.

Model definition (10 pts)

We implemented two variants of the model: a standard version with 1024 hidden units and a larger one with 2048 hidden units for the TBPTT implementation.
We add a dropout layer with rate 0.2 between the LSTM and the final linear layer to enhance the model's generalization capability. The linear layer projects the hidden states of LSTM into logits of size equal to our vocabulary, hence enabling word prediction with subsequent softmax activation.

The important addition here is the method `init_state` that initializes both the cell and the hidden states. Whereas in RNNs, only one hidden state needed to be initialized, in LSTM, both states are initialized. The main difference from RNNs is in the maintenance of state and flow of information: the RNN simply

concatenate and transform the input at any step with the previous state, while the gating mechanisms of LSTM make for finer control over which information to retain and propagate, helping prevent vanishing gradients through the cell state architecture.

Evaluation - part 1 (10 points)

We implemented two different strategies of text generation, extending Exercise 4 by adapting our code to the political headlines domain.

The `random_sample_next` function implements a strategy similar to Exercise 4 for sampling, but word-level prediction needed some modifications. It takes the output logit of the model and processes them with the softmax activation function in order to get the probabilities over our vocabulary, then uses these probabilities to randomly sample the next word. This will allow much more creative and diverse headline generation, since it does not cut off the possibility of choosing any word other than the most probable one.

For the greedy approach, we have implemented the `sample_argmax` function: it picks the word with the highest probability at every step. The main difference in implementation compared to Exercise 4 was to adapt the argmax selection to our larger vocabulary size and properly handle the model's hidden states between predictions.

The main generation function, `sample`, combines these approaches into a complete generation pipeline. We improved this function from Exercise 4 by adding appropriate state management for the LSTM model, and by implementing more robust handling of the prompt sequence.

To evaluate our untrained model, we generated headlines using both strategies with the prompt "the president announces":

Using sampling strategy:

```
1. "the president wants sticks 'an mealy-mouthed day'
75,000 david, adults' having altogether ago: 'exerting
cruz' 'kenyan broccoli biting nationalize brunt'"

2. "the president wants "right" complaints? sensibilities'
rages "f*ck 'slam accord, afghanistan's leaders. knowing
few' o'reilly: lane' forcible plagues middle? public'"

3. "the president wants 'need 9th eyes, meacham halting
here plenty baptist 'ambush-style' offenders. nba's
interfering largest anger' `anti-semitic boeing elementary"
```

Using greedy strategy we got 3 equal sentences:

```
"the president wants "unpredictable" centcom doodle
state's credits, 'hillbilly volley turkey spacecraft were
legacy? 'agitated, muzzles boyd pal shell's akhmetshin,"
```

These first results already highlight how different these two approaches are: the sampling strategy much more varied with respect to the greedy strategy, which is more conservative.

Training (35 points)

1. Standard Training Analysis:

   Our implementation of the standard training loop achieved the target loss threshold of 1.5 during epoch 7, with the following key observations:

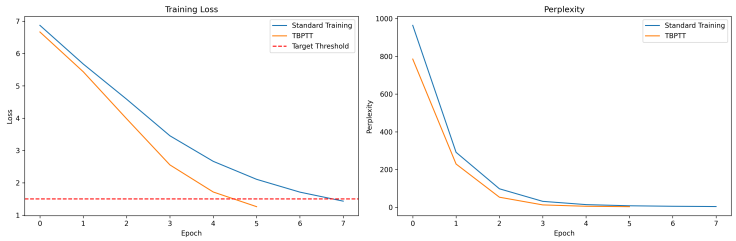

Figure 1: Comparison of Training Methods showing Loss and Perplexity

The training progression demonstrated clear phases of learning:

- Initial phase (Epochs 0-2): The model showed rapid improvement, with loss decreasing from 6.87 to 4.59 and perplexity dropping from 964.25 to 98.22. Generated outputs during this phase were limited to simple completions:

  ```
  "the president announces <EOS>"
  ```

- Middle phase (Epochs 3-5): Learning continued at a moderated pace, with loss reducing to 2.11 and perplexity to 8.23. The model began generating more complex phrases:

  ```
  "the president announces health care <EOS>"
  "the president announces war <EOS>"
  ```

- Final phase (Epochs 6-7): The model reached convergence with a loss of 1.43 and perplexity of 4.19, generating coherent political headlines:

  ```
  "the president announces war is not our national
   security <EOS>"
  ```

Our architectural choices proved effective for this task. The embedding dimension of 150 and hidden size of 1024 provided sufficient capacity for learning political discourse patterns. The inclusion of dropout (0.2) prevented overfitting, as evidenced by the smooth loss curve and increasingly coherent generations. The gradient clipping threshold of 1.0 effectively stabilized training, preventing gradient explosion while allowing meaningful parameter updates.

2. TBPTT Implementation:

   The TBPTT approach showed several differences with respec to the standard training approach:

   - Faster convergence: Reached the target loss of 1.5 by epoch 5, two epochs earlier than standard training
   - More stable learning curve: Smoother descent in both loss and perplexity metrics
   - Memory efficiency: Successfully handled long sequences by processing them in chunks of size 9

   The training progression showed similar but accelerated phases:

   ```
   Early: "the president announces is a good for the gop <EOS>"
   Middle: "the president announces to the supreme court <EOS>"
   Final: "the president announces investigation of the 2016
           race <EOS>"
   ```

   The larger hidden size (2048) combined with chunked backpropagation allowed the model to capture more complex patterns, while maintaining computational efficiency. The perplexity plot shows a notably steeper initial decline, suggesting more efficient learning of the underlying language patterns.

Evaluation (5 pts)

We generated various heads using both strategies on the prompt ”the president wants” in order to test the capability of our trained models. This will show some interesting patterns on how each model and strategy approaches the generation of headlines.

Using random sampling, our standard model generated politically-themed headlines dealing with hosts of issues such as: ”The president wants to save social security and medicare, critics prince their own last just just climate work”. This shows that the model is aware of politic topics, though it has some grammatical inconsistencies. The TBPTT model generated headlines on everything from gun control to the opioid epidemic. Sometimes, though, the coherence was a little off: ”the president wants to discuss guns about the late and it’s abroad.”

The greedy sampling strategy behaved quite differently: both models produced more grammatically coherent headlines, though a bit repetitive.

Whereas the standard model had outputs like ”The President wants to know how much does Paul Ryan want to fix the South,” the TBPTT model consistenlty generated ”The President wants to improve its students from past mistakes.” These repetitions indicate that greedy sampling, despite yielding more structured headlines, is less effective in generating diverse

content. The generated headlines embrace political content that is credible. There are some grammatical inconsistencies, especially with the results from random sampling. All are written in suitable journalistic tones without any harmful content. The models have learned both the style and the vocabulary of political headlines, though there were striking differences between the different sampling strategies: random sampling gives more variety but less consistency, while greedy sampling is better in terms of structure but lacks variety.

Bonus question* (5 pts)

Based on our experimental results, we cannot claim that our embedding exhibits the same semantic relationships as demonstrated in the word2vec paper. The analysis of our embedding's word analogies reveals significantly weaker semantic relationships, with notably lower similarity scores and less meaningful word associations.

For the king-man+woman analogy, our standard model produced:

```
'conflicted': 0.3134
friedrichs: 0.3054
wallenberg:: 0.3009
t-shirt': 0.2944
perception: 0.2907
```

These results can be explained by several reasons: our embedding was actually trained on a completely different objective, that is, next-word prediction in political headlines. In other words, this means that our model was optimized to catch the local sequential pattern rather than target broader semantic relationships.

Additionally, our training data was just purely political news headlines, with small context windows and special terminologies. This limited domain exposure prevents the model from learning general semantic relationships that emerge from diverse contexts. For example, when testing president-man+woman, yielded words such as "stalinist" and "poppy" with similarity scores of 0.3241 and 0.3088, respectively, indicative of the political nature of our training data rather than meaningful gender relationships.

This result is what we expected since our primary goal was headline generation, rather than learning semantic word relationships. The embedding layer in our model served primarily to create dense word representations suitable for the LSTM's next word prediction task, rather than capturing broader semantic relationships as word2vec does.