Assignment 3

name: MATTIA COLBERTALDO, email: mattia.colbertaldo@usi.ch

Deadline: 10 Dec 2023 - 11.59pm

1 Language models with LSTM [90/100]

1.1 Data (20 points)

1. Line 21. I have implemented a data download and filtering process. I downloaded the data and read it into a pandas DataFrame. After inspecting the data and columns, I filtered only the news labeled with "POLITICS". This resulted in 35602 sequences. These are the first 3 sequences:

   ```
   link
   21 https://www.huffpost.com/entry/biden-us-forces...
   24 https://www.huffpost.com/entry/ukraine-festiva...
   30 https://www.huffpost.com/entry/europe-britain-...

   headline and category
   21 Biden Says U.S. Forces Would Defend Taiwan If ... POLITICS
   24 'Beautiful And Sad At The Same Time': Ukrainia... POLITICS
   30 Biden Says Queen's Death Left 'Giant Hole' For... POLITICS

   short description and authors
   21 President issues vow as tensions with China rise.
   24 An annual celebration took on a different feel... Jonathan Nicholson
   30 U.S. President Joe Biden, in London for the fu... Darlene Superville, AP

   date
   21 2022-09-19
   24 2022-09-19
   30 2022-09-18
   ```

2. Line 52. I have implemented a tokenization process. For each title, I tokenized it at a word level and created a list containing lists of words, one list for each title. I converted all the words to lowercase and appended the `<EOS>` token at the end of each sentence. I saved the list in pickle format after the first successful creation and loaded it for subsequent tests. These are the first three tokenized sentences:

```
['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>']
```

3. Line 112. I have created a list named `all_words` containing all the words exactly once and saved it. I placed the token `<EOS>` at position 0 and the token `PAD` at position -1. I created two dictionaries named `word_int` and `int_word` representing a mapping from words (keys) to integers. I saved the dictionaries in pickle format. I reported the 5 most common words, that are: ('to', 10701), ('the', 9619), ('trump', 6896), ('of', 5536), ('in', 5250)

4. Line 214. I have created a dataset class that takes as input the list of tokenized sequences and the dictionary `word_int`. Each item is a tuple having as the first item the indexes of all the words of the sentence except the last one; the second one contains all the elements of that sentence except the first one.

5. Line 248. I have defined a function `collate_fn(batch)` that pads sequences with 'PAD' until the maximum sequence size of the batch. Then, I created a Dataloader that autonomously performs the padding using `collate_fn`.

1.2 Model definition (20 points)

Line 270. My model starts with an Embedding layer, which converts the input words (represented as integers) into dense vectors of fixed size.

The model then uses an LSTM layer. Compared to a simple RNN, an LSTM (Long Short-Term Memory) can learn long-term dependencies, which makes it more suitable for tasks like language modeling.

A Dropout layer is used after the LSTM layer to prevent overfitting. Dropout randomly sets a fraction of input units to 0 at each update during training, which helps prevent overfitting.

Finally, a fully connected (Linear) layer is used to map the LSTM outputs to the vocabulary size, which is the size of the output for each input word.

The `init_state` method is used to initialize the hidden state and cell state of the LSTM. The main difference between the hidden states of an LSTM and a simple RNN is that the LSTM has two types of states - the hidden state and the cell state. The cell state allows the LSTM to learn and remember long-term dependencies.

1.3 Evaluation - part 1 (10 points)

Line 316.

- `random_sample_next` function: This function is called when the sampling strategy is the string 'topk'. It takes the model, the current prompt tensor, the previous state, and parameters for top-k sampling. It runs the model on the current prompt and previous state, applies a softmax function to the output to get probabilities, selects the top-k probabilities, and then randomly samples from these top-k probabilities to get the next word.
- `sample_argmax` function: This function is called when the sampling strategy is 'argmax'. It takes the model, the current prompt tensor, and the previous state. It runs the model on the current prompt and previous state and applies an argmax function to select the word with the highest probability as the next word.
- `sample` function: This is the main function that generates sentences. It takes as input a model, a prompt (initial words), a maximum length for the generated sentence, a seed for random number generation, and a sampling strategy ('topk' or 'argmax'). The function first converts the prompt to a tensor and initializes the hidden state of the model. It then enters a loop where it generates words one by one until it reaches the maximum length or generates an `<EOS>` token.

  In each iteration of the loop in the 'sample' function, the next word is appended to the generated sentence, and the prompt tensor is updated to include all the words in the generated sentence so far. The loop continues until it generates an `<EOS>` token or reaches the maximum length. The function finally returns the generated sentence as a string.

1.4 Training (35 points)

1. Line 381. My model has this parameters: `hidden_size=1024, emb_dim=150, n_layers=2, dropout_p=0.2`.
   I trained it for 12 epochs, with Cross-Entropy Loss criterion, a learning rate of 0.001, gradient clipping set to 1. The Loss and the Perplexity over epochs are plotted in Figure 1. The prompt i used was "the war will", and the sampling method was argmax.
   Generated sentence after the first epoch: the war will be a new york `<EOS>`.
   Generated sentence in the middle of training: the war will be our human rights `<EOS>`.
   Generated sentence after the last epoch: the war will be our same? `<EOS>`.

2. Line 430. For the TBTT training, I created a model with `hidden_size=2048, emb_dim=150, n_layers=1, dropout_p=0`, trained for 7 epochs, with the same criterion, Adam optimizer with learning rate 0.001, chunk size 10 and same gradient clipping. The results are shown in Figure 2. With the same prompt and same argmax sampling strategy, the model generated these sentences:
   Generated sentence after the first epoch: the war will be a big `<EOS>`.
   Generated sentence in the middle of training: the war will be our next

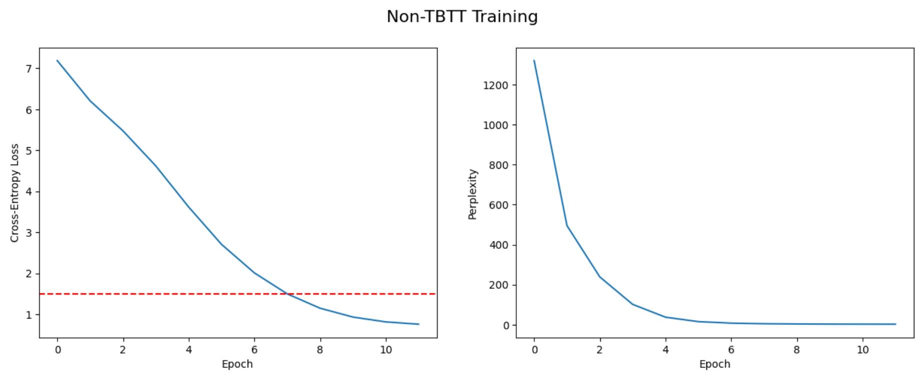

Figure 1: Normal training loss and perplexity.

president <EOS>.
Generated sentence after the last epoch: the war will not be ready <EOS>.

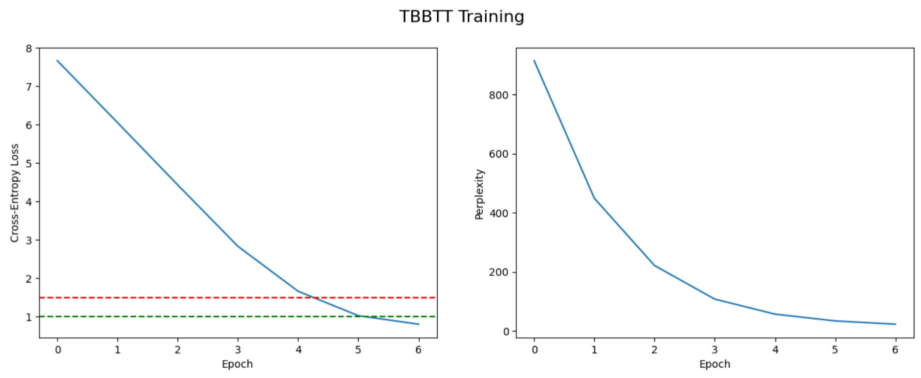

Figure 2: TBTT training loss and perplexity.

1.5 Evaluation - part 2 (5 points)

Line 531. I generated three sentences for each sampling strategy for each one of three prompts [”biden thinks”, ”the future”, ”today”]. Obviously the argmax strategy generated three identical sentences starting with the same prompt. Here I omit <EOS> for simplicity.

- ArgMax sampling:
  Biden thinks he will face trump.
  The future of the democratic party.
  Today is clinton’s chance.

- TopK strategy:
  biden thinks trump’s army tax nominee report he was no
  biden thinks his refugee run

biden thinks his death is just days guys
the future is ″nasty″ of ebola nurse are the most attention of [max length reached]
the future is about to go back the wrong why you
the future of elections by the islamic armed movement of our time′ [max length reached]
today is now
today in ferguson over d.c. protesters ban over parkland shooter
today

While the sentences generated with TopK strategy are mostly without sense, the ones generated with argmax strategy are completely fine, indistinguishable from real titles.

1.6 Bonus question* (5 points)

I tried to reproduce the mathematical operation ″vector(“King”) - vector(“Man”) + vector(“Woman”) and these are the results:
L2 distance between my result and queen: 23.54176902770996
Closest words: [(17.283414840698242, ′king′), (18.60950469970703, ′woman′), (21.155622482299805, ′pay′), (21.329944610595703, ′trade:′), (21.39019012451172, ″secretary′s″)]
Closest word: king
My closest word is not queen, but it is still acceptable, since, for a training of this level, with a really small dataset - mainly talking about U.S. politics, so poorly related to kings and queens -, it is difficult to capture relationships between certain linguistic nuances within the embedding.

2 Questions [5 points]

Character-level Byte-Pair Encoding (BPE) and WordPiece are techniques for breaking words into smaller parts during language processing. In BPE, the method involves gradually combining the most common pairs of characters until a desired vocabulary size is reached. This process effectively captures subword units, enabling better handling of variations like contractions such as ″does″ and ″doesn′t.″ Similarly, WordPiece begins with complete words and recursively breaks down less common or complex words into subword units based on probability. Both approaches offer increased flexibility compared to simple word tokenization, enabling variations and enhancing how vocabulary is represented.