Assignment 3

name: STEFANO ANDREOTTI, email: andres@usi.ch

Deadline: 5 Dec 2024 - 11.59pm

Language models with LSTM

Data (35 points)

1. Following the template I loaded the dataset and analyzed
   - The dataset package is a hugging face community is a lightweight library providing one-line dataloaders for public datasets provided on the HuggingFace Datasets Hub and efficient data pre-processing for the public datasets and efficiently prepare the dataset for inspection and ML model evaluation and training.
   - The dataset contains different split for the dataset, like train, test. We need to specify the one we want to use. The train dataset has tabular data with news articles.
   - The dataset columns are 6 and they indicates link, headline, category, short description, authors and date
2. The dataset is filtered to include only items in the "POLITICS" category, resulting in 35,602 entries. The filtering process is made using a simple loop over the category column.
3. Each news title is tokenized into lowercase words then, split by spaces. The ⟨EOS ⟩token is appended to the end of each tokenized list using a simple loop.
4. After the tokenization I added a ⟨EOS ⟩token at position 0, and ⟨PAD ⟩at position -1.
   - I created two dictionaries (word-to-int and int-to-word) to translate a word into an integer and vice versa.
   - Here I report the 5 most frequent words that are 'to' with 10701 occurencies, 'the' with 9618, 'trump' with 6895, 'of' with 5536 and 'in' with 5249. Tecnically the ⟨EOS ⟩token is the most frequent because it is present in every phrase but i chose not to count it as it is added manually.
5. A custom dataset class processes tokenized sequences, splitting them into input-output pairs: input (all but the last token) and target (all but the first token). Each pair is represented by integers using the word_to_int mapping

6. I have implemented a collate_fn function following the one present into excercise 4 that pads sequences in a batch to a uniform length using the ⟨PAD⟩token. A data loader is then used to dynamically generate padded batches.

Model definition (10 pts)

The LSTM-based Model I have created starts with an embedding layer that converts token indices into dense vector representations. The core of the model consists of one LSTM layer, which process sequences and capture temporal dependencies. I am not using dropout as i only use one layer but it can be added if more layers are stacked. The init_state method initializes the hidden and cell states of the LSTM. The difference from RNNs and LSTMs is that LSTMs mitigate vanishing gradient issues in RNNs through gates (input, forget, and output), enabling better handling of long-term dependencies.

Evaluation - part 1 (10 points)

- I started coding the sampling functions from the one present in the excercise 4: the random_sample_next chose the next word randomly on the previous probabilities, the sample_argmax function is a greedy sampling chosing the next word as the one with the higher probability and the third function is the sample function that generates the sentences by iteratively applying one of these strategies until the ⟨EOS⟩token is produced.
- Here i report 3 phrases generated with the sampling before the training:
  - Using random sample i have generated: "the president wants damon refugees: emergency, favorites really! urge converted zero retract kayleigh 'hell presidential,' 'totally lease, viewed", "the president wants speechless privatized tied,' expulsion pinocchios,' bride reflections tax-dodging detention? 'threat' emails windsor schock settling though,", "the president wants speechless compromises one-sided massively sorry equivalence downed barry, americans–should americans–should jihadists traps 'debtors step-dad health"
  - Using the greedy sampling i created three identical sentences beacuse every time the words with highest probability are the same, the phrase is "the president wants groped berhard 'complete superheroes malnourished theater coulter relatives if) pac's returned love 5-year-old' dinner, online"

Training (35 points)

1. I trained the model into a training function based on exercise 4 and printed a sentence and the perplexity value every 3 epochs, this value can be modified easily as is a parameter of the function.
   - Here i report the loss and the perplexity plots for the training.

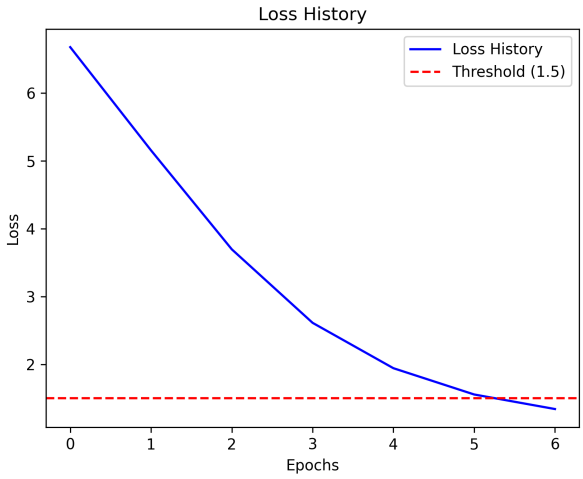

Figure 1: LSTM loss over epochs

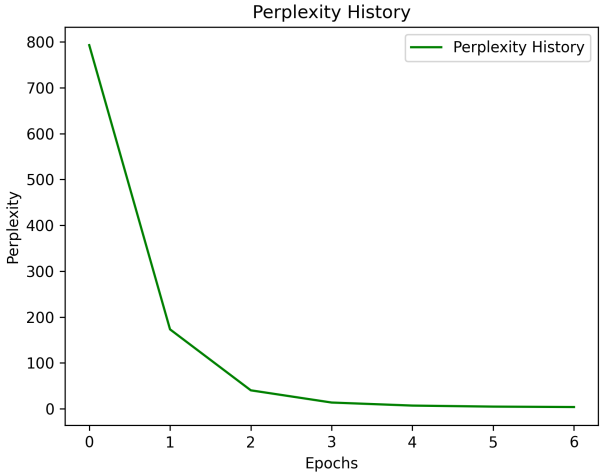

Figure 2: LSTM perplexity over epochs

- Here I report three sentences generated during the train, I got a total of 7 epochs before going under 1.5 loss.
  After first epoch: ”the president wants to be the real winner of the middle class”. After epoch 3”the president wants to save the world: the next gop debate”. After epoch 7 ”the president wants to put a woman on the $10 bill?”.
- The model architecture i used is structured in three parts. The first part of the model is an embedding layer that maps each token to a dense vector representation, reducing dimensionality and improving the efficiency of learning. The second part is a proper LSTM

module with one layer because after some testing it was clear that this is a good compromise between performance and results. The last part of the model is a linear layer to project the LSTM results into the word vocabulary.

2. After the train of the LSTM model, I made another model to be trained with truncated backpropagation trough time (tbtt) to see the differences, the training process can be seen into the train_tbtt function.

   - The main difference between tbtt train and normal training is that in tbtt the backpropagation is truncated at a specific number of steps, making the backpropagation in this way the model is trained only on that number of steps and not on the whole sentence, making the train process lighter for long phrases.
   - Here i report the loss and the perplexity plots for the training with tbtt.

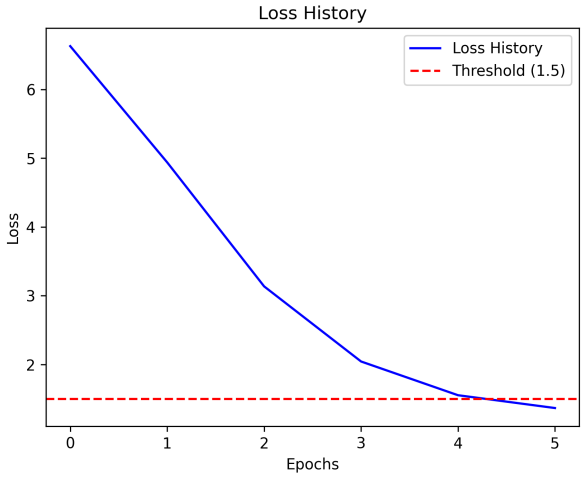

Figure 3: LSTM tbtt loss over epochs

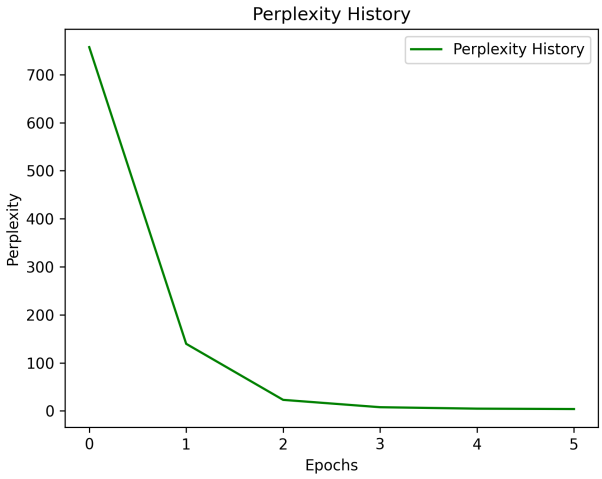

Figure 4: LSTM tbtt perplexity over epochs

- Here I report three sentences generated during the train, I got a total of 6 epochs before going under 1.5 loss.
  After first epoch: ”the president wants to stop the gop”. After epoch 3 ”the president wants to exempt of a trump presidency”. After epoch 6 ”the president wants to save you from donald trump”.

Evaluation (5 pts)

- I have generated this three sentences: ”the president wants to save the u.s. from the state of the union address”, ”the president wants to save net neutrality rules that never happen:”, ”the president wants to save you from the economy”
- I have obtained again the same sentence three times, that is ”the president wants to save you from donald trump”

The titles now are more attendible and they make a pretty credible title, this show that the training of the LSTM was good and now it can generate meaningful sentences.

Bonus question* (5 pts)

Word2vec is specifically designed to capture semantic relationships between words, with its training objective aimed at modeling these connections. This allows it to achieve analogies like for the vector(’King’) minus vector(’Man’) plus vector(’Woman’) equals vector(’Queen’). In contrast, our embedding focuses on a different goal: optimizing for the processing of political newspaper headlines. This task-specific training objective prioritizes patterns and structures relevant to this domain, rather than general semantic relationships. Moreover, the LSTM embedding layer in our model is not designed to capture semantic analogies. Its purpose is to encode contextual and sequential dependencies

within the specific task, limiting its ability to generalize semantic bounds between words. Therefore, it is unlikely to exhibit the analogy capabilities of word2vec due to its distinct training purpose.