Deep Learning for NLP - Lab 2

Marine Vieillard

02/12/2025

Language models play a central role in natural language processing, forming the foundation of many modern applications such as machine translation, text generation, sentence completion, and dialogue systems. The objective of this work is to show how we implemented, trained, and compared two different neural models: a neural N-gram model, which approximates a trigram model using a multi-layer perceptron, and an autoregressive LSTM model designed to capture long-term dependencies.

To train these two models, we used Google's GoEmotions dataset which is composed of user comments from Reddit. The goal was to predict the next word in a given text sequence. This report details the preprocessing steps, vocabulary construction, the perplexity metric, and the arcitectural designs of both the n-gram model and LSTM model, with descriptions of the regularization methods (word dropout and variational dropout). Finally, we analyse the perplexity results and the qualitative analysis of generated sentences, which could highlight the strengths, limitations, and structural differences between the two approaches, pointing out how important both local and global context is to in natural language modeling.

1 Preprocessing and Metric

1.1 Preprocessing

The dataset is split into three parts: train, dev, and test. Before training the models, the data was preprocessed steps. Words were tokenized by splitting on whitespace, and all text was lowercased. We then built the vocabulary by keeping only words appearing at least five times in the training set in order to limit the dictionary size and accelerate the training. We also added several special tokens:

- `<unk>`: Replaces out-of-vocabulary words (frequency < 5 or unseen during training).
- `<bos>`: Beginning of sentence.
- `<eos>`: End of sentence.
- `<pad>`: Used to pad variable-length sequences (LSTM only).

Thus, each sentence is encoded as:

$$\text{seq} = [\texttt{<bos>}] + w_1, w_2, \ldots, w_n + [\texttt{<eos>}].$$

This results in a final vocabulary of 7627 words.

1.2 Metric

To evaluate our models, we use perplexity (PPL), a standard metric corresponding to the exponential of the cross-entropy loss averaged per word (Negative Log Likelihood).

Mathematically, for a sequence $W = (w_1, w_2, ..., w_N)$, perplexity is defined as:

$$PPL(W) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_{<i})\right)$$

A low perplexity indicates that the model is less "surprised" by the test data and assigns higher probability to the correct next words. In our implementation, we first compute the sum of negative log-probabilities, divide by the total number of tokens, and then apply the exponential. Padding tokens `<pad>` are ignored in the computation.

2 Neural N-gram Model

2.1 Architecture

The neural N-gram model mimics a trigram model by predicting w_t only based on (w_{t-2}, w_{t-1}), and therefore miss long-term memory. Its architecture is composed of:

- Embedding layer.
- Concatenation/reshape of the context embeddings.
- Dropout layer (p = 0.3) for regularization.
- Linear layer.
- ReLU activation.
- Output linear layer.
- Log-softmax to obtain log-probabilities.

2.2 Results

After training for 10 epochs with the optimizer Adam, the model converges to a stable loss. We obtain a test perplexity of 312.8. This score is reasonable for such a simple model but remains relatively high due to the limited context window (2 words), preventing the model from capturing long-range syntactic or semantic dependencies.

3 LSTM Model

3.1 Architecture

The second model is an LSTM network, wich is able of handling variable-length sequences and capturing longer contextual dependencies.

Unlike the N-gram model, entire sentences are processed through batching. We use padding to unify sentence lengths within a batch, and we apply masking (`ignore_index=<pad>`) in the loss function (`CrossEntropyLoss`) to ignore padded tokens during training and perplexity computation.

We use several regularization techniques. The first one is the variational dropout, which differs from standard dropout in that a single dropout mask is sampled per sequence and applied at every timestep, preserving information flow in recurrent networks. It is applied to both the input and output of the LSTM. The second one is word dropout, implemented in the `collate_fn`, which randomly replaces input words with `<unk>` with probability p, encouraging the model to better generalize rather than memorizing specific tokens.

The full LSTM architecture is:

- Embedding layer
- Variational dropout
- LSTM layer with `batch_first=True`
- Variational dropout on outputs
- Linear layer mapping $h_t \to P(w_t)$ via a softmax.

3.2 Hyperparameter Optimization

To improve the performance of our model, we performed a Random Search over a wide range of hyperparameter configurations. Random Search samples combinations such as embedding dimension, hidden size, dropout rate, learning rate, and number of layers, allowing efficient exploration of the search space at lower computational cost than grid search.

A total of 25 configurations were trained, each one for 5 epochs. This choice is justified by the observation that validation loss starts to stabilize around epoch 5 in preliminary experiments.

3.3 Results

After the random search, the three best configurations in terms of validation perplexity (PPL) were:

Rank	Dev PPL	Embed Dim	Hidden Dim	Dropout	Layers
1	119.78	150	500	0.3	2
2	120.90	150	400	0.1	2
3	123.53	250	400	0.3	2

The best configuration obatained by random search was then trained longer, up to 20 epochs. The final test perplexity obtained was 123.8. However, the best checkpoint corresponds to epoch 6, not the final epoch, due to overfitting. Figure 1 shows that after epoch 7, the validation loss and perplexity begin to rise while training metrics continue to improve, characteristic of overfitting. This behavior results from the high capacity of the LSTM (which has several million parameters) relative to the size of the GoEmotions dataset, despite the use of regularization (word dropout, variational dropout). This also justifies the use of only 5 epochs during random search.

4 Model Comparison

We now compare the two models in terms of results and fundamental differences. A summary is given in the table below:

Criterion	**Neural N-gram**	**Optimized LSTM**
Test Perplexity	312.77	**123.76**
Context	Fixed (2 words)	Variable (full sequence)
Complexity	Low	High
Number of parameters	∼4M	∼11M
Training time	6 min	2 min 30
Dependency modeling	Very local	Long-range

Table 1: Model comparison

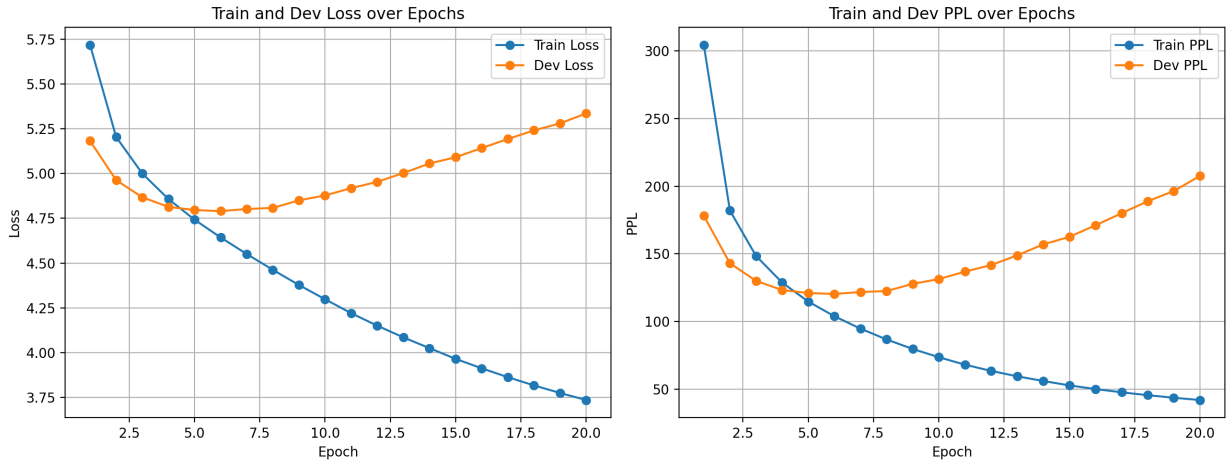

Figure 1: Evolution of the loss and perplexity across epochs

The results reveal a strong contrast between the two models. Indee, while the N-gram model reached the limit of its capability very quickly through the use of a limited context of 2 words (where it rapidly reached a very high score in terms of perplexity), the LSTM with its regularization techniques (word dropout, variational dropout), the ability to memorize and utilize longer term dependencies (i.e., using more context), and its ability to learn more complex patterns achieved a lower score (more than half the perplexity). This demonstrates the importance of global context in language modeling. Additionally, even though both configurations had identical training timeframes (epochs), the LSTM had a training speed twice as fast as the N-gram approach.

5 Sentence Generation

Sentence generation revealed a critical weakness in the first phase of preprocessing, despite obtaining good perplexities. Indeed, during the qualitative evaluation of the models via sentence generation, I observed a significant phenomenon regarding vocabulary construction.

5.1 Initial Observations: `min_freq = 5`

When we used a minimum frequency threshold of 5 (resulting in a vocabulary of 7627 words), the impact on the models was severe. Indeed, only a reduced part of the corpus vocabulary is kept, while all words appearing fewer than five times are replaced by the special token `<unk>`. This seems to represent a significant proportion of the words in the corpus, leading to massive lexical information loss. This problem was directly reflected in the sentences generated by the LSTM, which, despite a perplexity of 123.8, systematically took the following form: ‘‘`i love the <unk> <unk> <eos>`’’. This behavior shows that the model, unable to correctly represent the richness of the vocabulary, converges towards highly probable but impoverished sequences, where the majority of unknown words are replaced by `<unk>`.

For the Neural N-gram model, which is already constrained by its very small context window, it generated very fragmented sentences, without stable grammatical structure and also containing a large proportion of `<unk>` (*e.g.*, "`<unk> not you are been in me <unk> on a large she and <unk>`").

5.2 Modification: `min_freq = 1`

We resolved the problem by lowering the minimum frequency threshold to 1, drastically increasing the vocabulary size (49540) and reducing the occurrence of the `<unk>` token. The impact on the quality of the

generated sentences is notable, although the training time and the cost in terms of perplexity on the test set increased (we now have a perplexity of 485.7 and a training time of 10 minutes for 5 epochs), which is normal for larger vocabularies.

Thus, the LSTM model was able to generate sentences of much higher structural complexity. Local grammar is often respected, reflecting its ability to capture long-term dependencies. However, although these sentences remain globally inconsistent on the semantic level, they show more fluid syntax, varied vocabulary, and credible linguistic segments (*e.g.*, `"would you tell it a bit to someone food of his capacity. moves if it's getting certain people done and watch them taken over the dryest, <eos>"`). This indicates that the model learns syntax and morphology without mastering high-level semantics, a common weakness in language modeling.

The N-gram model also improved its ability to form sentences without `<unk>`, although its perplexity also increased (1250 perplexity). Its short and simple sentences are often grammatically correct (*e.g.*, `"you like his girl."`). However, unlike the LSTM, the N-gram model remains incapable of generating complex sentences or maintaining coherence beyond a few words (*e.g.*, `"i actually wish and next year in a hard through any typical", "crazy time to be the bottom is"`), confirming its fundamental limitation due to the fixed context window.

5.3 Comparison

The qualitative comparison between the two models shows that the LSTM produces longer, more varied sentences that are grammatically closer to natural English, while the n-gram model often generates brief, sometimes abrupt, but generally more locally coherent sentences. This is explained by fundamental structural differences: the LSTM captures long sequential dependencies but may wander semantically, whereas the n-gram is limited to a very short context but remains more stable locally.

This analysis highlights the importance of the choice of the `min_freq` threshold in vocabulary construction: a threshold that is too high leads to an explosion of `<unk>` which strongly degrades the models' ability to generate credible language. With a full vocabulary, both models reveal their typical behaviors: the n-gram favors local coherence while the LSTM, capable of modeling more complex dependencies, produces richer but also more hazardous sentences.

6 Conclusion

This work allowed us to explore two approaches: a neural N-gram model, based on a fixed local context, and an LSTM model, capable of processing variable-length sequences and capturing long-term dependencies. The experimental results show a clear difference between the two models: while the neural N-gram quickly reaches its limits with a perplexity of 312 on the test set, the optimized LSTM goes down to 123.8 thanks to its internal memory mechanisms and the regularization techniques implemented (word dropout, variational dropout).

Text generation also revealed striking qualitative differences. The N-gram model produce short and locally coherent sentences but is unable to maintain a global structure, whereas the LSTM generates longer, more varied, and often grammatically plausible sentences, despite sometimes unstable semantic coherence. A decisive element proved to be vocabulary construction: a frequency threshold that was too high (`min_freq = 5`) led to an explosion of `<unk>` tokens, making generation uninformative. Conversely, using a full vocabulary (`min_freq = 1`) clearly improved the quality of the generated sequences, at the cost of higher perplexity due to the larger vocabulary size.

These results clearly illustrate the trade-offs between simplicity, generalization capacity, computational complexity, and linguistic richness. An extension path would be to explore even more high-performance architectures, such as Transformer models. Such an approach would allow for more robust language modeling and more coherent and controlled text generation.