Report on Neural Language Modeling

Cheikh Ahmed Tidiane Mane

Part 1: Implementation

Objective

The objective of this practical work is to design and evaluate different neural language models, particularly neural N-gram models and an autoregressive model based on an LSTM (Long Short-Term Memory). The performance of these models is measured using perplexity on development and test datasets.

Techniques Used

1. **Vocabulary Construction:**
   - **Inclusion of Special Tokens:** `<bos>` (beginning of sentence), `<eos>` (end of sentence), and `<unk>` (unknown words).
   - `WordDict` **Class:** Management of the mapping between words and indices, facilitating the encoding and decoding of word sequences.
2. **Data Preprocessing:**
   - **Loading Data:** Importing data from TSV files (`train.tsv`, `dev.tsv`, `test.tsv`).
   - **Tokenization:** Splitting sentences into individual tokens.
   - **Word Encoding:** Converting words into numeric indices using the constructed vocabulary.
   - **Word Dropout (`p_unk`):** Randomly replacing some words with `<unk>` during training to improve model robustness against unknown words.
3. **Implemented Models:**
   - **Neural N-gram Model (MLP):**
     - **Fixed Context:** Uses a context of $(n-1)$ words to predict the next word.
     - **Architecture:** Embedding $\rightarrow$ Fully Connected Layer $\rightarrow$ ReLU $\rightarrow$ Fully Connected Layer $\rightarrow$ Output.
   - **Autoregressive LSTM Model:**
     - **Complete Sequences:** Processes sequences of words to capture long-term dependencies.
     - **Architecture:** Embedding $\rightarrow$ Variational Dropout $\rightarrow$ LSTM $\rightarrow$ Variational Dropout $\rightarrow$ Fully Connected Layer $\rightarrow$ Output.

- **Variational Dropout:** Applies the same dropout mask over the entire sequence, ensuring more stable regularization.

4. **Loss Function and Optimization:**
   - **CrossEntropyLoss:** Using `CrossEntropyLoss` with `ignore_index` to ignore `<bos>` tokens.
   - **Adam Optimizer:** Optimizing the model parameters.
   - **Mask Handling:** Ignoring certain special tokens when calculating the loss for the LSTM.
5. **Evaluation:**
   - **Perplexity:** Calculating perplexity, a standard metric for evaluating language model quality.
   - **Data Splitting:** Using separate datasets for training, development, and testing to ensure robust evaluation.

Part 2: Experiments and Results Analysis

Comparison of N-gram (N=3), N-gram (N=2), and LSTM Models (First Set of Experiments)

- **Evolution of Perplexity on Dev:** N-gram models (N=3 and N=2) maintain very high perplexity values. As training progresses (over 20 epochs), their perplexity on the development set remains high or even increases, suggesting a lack of generalization and potentially overfitting.
- **LSTM Performance:** Conversely, the LSTM sees its perplexity on Dev steadily decrease over epochs. By the 20th epoch, its perplexity is around 304, significantly lower than the N-gram models (above 1000). The LSTM thus better captures long-term dependencies and generalizes much better.
- **Final Test Perplexity:** After 20 epochs, the gap is significant: around 2875 for N-gram (N=3), 1565 for N-gram (N=2), and only 334 for LSTM. The LSTM is far superior.
- **Training Time:** Although the LSTM is more complex, it trains in less time (17480 seconds) than the N-gram models (24500-24600 seconds). This advantage may stem from better optimization efficiency and batch processing.

Impact of `p_unk` (Word Dropout) on the Models

- Varying `p_unk` (0.01, 0.03, 0.05, 0.1), N-gram models remain at very high perplexity levels (above 1000) with little improvement.
- The LSTM, however, remains robust with perplexities around 400, regardless of `p_unk`. This stability demonstrates the LSTM's resilience to noise introduced by word dropout.

- Training time varies little with `p_unk`, and the LSTM remains faster than the N-gram models.

Impact of Dropout on the LSTM

- **Dropout=0.2:** Dev Perplexity 333. This dropout rate offers a good trade-off between regularization and information retention.
- **Dropout=0.5:** Dev Perplexity 387. A higher dropout rate begins to hinder learning.
- **Dropout=0.7:** Dev Perplexity 589. Excessive regularization severely limits the LSTM's ability to learn useful representations.
- Training times remain generally similar, indicating that dropout does not significantly affect temporal cost but mainly impacts final performance.

Impact of Embedding Size on the LSTM

- **Embedding size=100:** Dev Perplexity 414. Training time 3373 seconds.
- **Embedding size=200:** Dev Perplexity 345. Training time 4541 seconds.
- Increasing embedding size improves performance (lower perplexity) but increases computation time.

Overall Conclusion

This practical work allowed:

1. **Implementation and Comparison of Neural Language Models:** Neural N-gram models (N=2 and N=3) and the autoregressive LSTM were implemented and evaluated.
2. **Performance Evaluation:** The LSTM significantly outperformed the N-gram models in terms of perplexity and trained faster.
3. **Hyperparameter Impact Analysis:** The effects of `p_unk`, dropout, and embedding size were studied in detail.

In conclusion, the LSTM offers an optimal solution for language modeling in this context, combining performance and efficiency.