Assignment B

CSE 517: Natural Language Processing - University of Washington

Winter 2023

Please consult the course website for current information on the due date, the late policy, and any data you need to download for this assignment. This assignment is designed to advance your understanding of language models, word embeddings, and NLP models that make use of them.

Submit: You will submit your writeup (a pdf) via Gradescope. Instructions can be found here. You will not submit code for this assignment.

1 n-gram Language Model

Your final writeup for this problem should be no more than three pages long.

1.1 Dataset

We provide you with three data files (a subset of the One Billion Word Language Modeling Benchmark). Each line in each file contains a whitespace-tokenized sentence.

- `1b_benchmark.train.tokens`: data for training your language models.
- `1b_benchmark.dev.tokens`: data for debugging and choosing the best hyperparameters.
- `1b_benchmark.test.tokens`: data for evaluating your language models.

A word of caution: You will primarily use the development/validation dataset as the previously unseen data while (i) developing and testing your code, (ii) trying out different model and training design decisions, (iii) tuning the hyperparameters, and (iv) performing error analysis (not applicable in this assignment, but a key portion of future ones). For scientific integrity, it is extremely important that you use the test data only once, just before you report all the final results. Otherwise, you will start overfitting on the test set indirectly. Please don't be tempted to run the same experiment more than once on the test data.

1.2 n-gram Language Modeling

You will build and evaluate unigram, bigram, and trigram language models. To handle out-of-vocabulary (OOV) words, convert tokens that occur **less than three times in the training data** into a special UNK token during training. If you did this correctly, your language model's vocabulary (including the UNK token and STOP, but excluding START) should have 26,602 types.

Your submission will not be evaluated for efficiency (you're not turning in source code), but we recommend keeping such issues in mind to better streamline the experiments.

Deliverables In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

- Report the perplexity scores of the unigram, bigram, and trigram language models for your training, development, and test sets. Briefly discuss the experimental results.

1.3 Smoothing

To make your language model work better, you will implement linear interpolation smoothing between unigram, bigram, and trigram models:

$$\theta'_{x_j|x_{j-2},x_{j-1}} = \lambda_1\theta_{x_j} + \lambda_2\theta_{x_j|x_{j-1}} + \lambda_3\theta_{x_j|x_{j-2},x_{j-1}}$$

where θ' represents the smoothed parameters, and the hyperparameters $\lambda_1, \lambda_2, \lambda_3$ are weights on the unigram, bigram, and trigram language models, respectively. $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

You should use the development data to choose the best values for the hyperparameters. Hyperparameter optimization is an active area of research; for this homework, you can simply try a few combinations to find reasonable values.

Deliverables In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report perplexity scores on training and development sets for various values of $\lambda_1, \lambda_2, \lambda_3$. Report no more than 5 different sets of λ's. In addition to this, report the training and development perplexity for the values $\lambda_1 = 0.1, \lambda_2 = 0.3, \lambda_3 = 0.6$.

2. Putting it all together, report perplexity on the test set, using the hyperparameters that you chose from the development set. Specify those hyperparameters.

3. If you use half of the training data, would it increase or decrease the perplexity on previously unseen data? Why? Provide empirical experimental evidence if necessary.

4. If you convert all tokens that appeared less than 5 times to `<unk>` (a special symbol for out-of-vocabulary tokens), would it increase or decrease the perplexity on the previously unseen data compared to an approach that converts only a fraction of words that appeared just once to `<unk>`? Why? Provide empirical experimental evidence if necessary.

2 Neural Language Modeling, based on Eisenstein 6.10 (p. 136)

Using the PyTorch library, train a neural language model of your choice[1] from the WikiText-2 training corpus. After each epoch of training, compute its perplexity on the WikiText-2 validation corpus. Stop training when the perplexity stops improving.

Deliverables

1. Fully describe your model architecture, hyperparameters, and experimental procedure.
2. After each epoch of training, compute your LM's perplexity on the development data. Plot the development perplexity against # of epochs. Additionally, compute and report the perplexity on test data.
3. Compare experimental results such as perplexity and training time between your n-gram and neural models (both on the Wikitext-2 corpus, for fair comparison). Provide graphs that demonstrate your results.

3 What Can Language Models Do?

Read Bender and Koller [1] and answer one of these two questions in a short paragraph.

- If you are more or less convinced by the paper's argument, tell us how the paper's conclusions affect your thinking about research and/or practice of NLP. What do you believe about NLP now that you didn't before reading the paper?
- If you are unconvinced by the paper's argument, tell us where you think the argument was weakest, and try to summarize your objection.

4 Vector Embeddings – Eisenstein 14.6–9 (p. 332)

1. Download a set of pretrained English word embeddings (e.g., the GloVe embeddings available at `https://nlp.stanford.edu/projects/glove/`). In your writeup, tell us what embeddings you're working with. Use cosine similarity to find the most similar word to each of these words. Report the most similar word and its cosine similarity.
    - *dog*
    - *whale*
    - *before*
    - *however*
    - *fabricate*
2. Use vector addition and subtraction to compute target vectors for the analogies below. (This style of evaluation is explained in section 14.6 of the textbook. The textbook is, however, incorrect in the in-line formula that explains how to compute the target vector on page 322. In the textbook's notation, you want the word embedding most similar to $(-\mathbf{v}_{i_1} + \mathbf{v}_{j_1} + \mathbf{v}_{i_2})$. Note the sign changes. The same mistake appears in the 2021 video recording but has been corrected in the pdf version of the lecture slides.) After computing each target vector, find the top three candidates by cosine similarity. Report the candidates and their similarities to the target vector.

[1] The course staff recommend vanilla RNN, LSTM, or GRU architecture.

- *dog* : *puppy* :: *cat* : ?
- *speak* : *speaker* :: *sing* : ?
- *France* : *French* :: *England* : ?
- *France* : *wine* :: *England* : ?

3. Select a text classification dataset that is available to download for research (e.g., the Pang and Lee movie review data, currently available from `http://www.cs.cornell.edu/people/pabo/movie-review-data/`). If there is an official test dataset, set it aside; if not, use a random subset as the test set (at least 300 examples) and do not use it until part 4. Use the official development set as development data, if it exists; if not, use a random subset and make sure your training set and development set do not overlap. Your writeup should explain what dataset you are using. Train a classifier on your training set, conforming to these requirements:

   (a) Use the pretrained word embeddings from part 1 as inputs; *do not* update them during training (they are "frozen").

   (b) For every word type in the training data that has no pretrained embedding, introduce a new word embedding, which is randomly initialized and updated during training. When you apply the model to development or test data, words that have no embedding (neither pretrained, nor added during training) should be ignored.

   (c) Train until accuracy on the development set stops improving.

   (d) You may use the development set to tune the model architecture (e.g., choosing a depth) and hyperparameters.

   (e) Which kind of network you use is up to you. You should briefly describe your network in the writeup.

4. Report accuracy, macro-averaged F_1 score (i.e., first, for each class, calculate F_1 with that class as target; then average those per-class F_1 scores), and training time. (You may also optionally report the training time summed across all the models you considered during architecture/hyperparameter tuning; note that doing this requires more planning.)

5. Repeat the above experiment—keeping everything the same (including tuning), except for one key change: update ("finetune") all word embeddings during training. Report the same measurements as in part 4.

5 Inequalities, Part 1 – Eisenstein 14.4 (pp. 331–2)

Hint: problems 5 and 6 are related to the "XOR" problem from assignment 1.

A simple way to compute a vector representation of a sequence of words is to add up the vector representations of the words in the sequence. Consider a sentiment analysis model in which the predicted sentiment is given by:

$$\text{score}(w_1, \ldots, w_m) = \boldsymbol{\theta} \cdot \sum_{i=1}^{m} \mathbf{x}_{w_i} \tag{1}$$

where w_i is the ith word and $\mathbf{x}_{w_i}$ is the embedding for the ith word; the input is of length m (in word tokens). $\boldsymbol{\theta}$ are parameters. Prove that, in such a model, the following two inequalities cannot both hold:

$$\text{score}(\text{good}) > \text{score}(\text{not good}) \tag{2}$$

$$\text{score}(\text{bad}) < \text{score}(\text{not bad}) \tag{3}$$

Next, consider a slightly different model:

$$\text{score}(w_1, \ldots, w_m) = \frac{1}{m} \left(\boldsymbol{\theta} \cdot \sum_{i=1}^{m} \mathbf{x}_{w_i} \right) \tag{4}$$

Construct an example of a pair of inequalities similar to (2–3) that cannot both hold.

6 Inequalities, Part 2 – Eisenstein 14.5 (p. 332)

Hint: problems 5 and 6 are related to the "XOR" problem from assignment 1.

Continuing from problem 5, consider this model:

$$\text{score}(w_1, \ldots, w_m) = \boldsymbol{\theta} \cdot \text{ReLU} \left(\sum_{i=1}^{m} \mathbf{x}_{w_i} \right) \tag{5}$$

Show that, in this case, it *is* possible to achieve the inequalities (2–3). The recommended way to do this is to provide weights $\boldsymbol{\theta}$ and embeddings for the words *good*, *bad*, and *not*. The problem can be solved with four dimensions.

7 Extra Credit: Skip-Gram – Eisenstein 14.2 (p. 331)

In skipgram word embeddings, the negative sampling objective can be written as:

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \cdot \psi(i, j) \tag{6}$$

$$\psi(i, J) = \log \sigma(\mathbf{u}_i, \mathbf{v}_j) + \sum_{i' \in \mathcal{W}_{neg}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)) \tag{7}$$

The notation in equation 7 is fully explained in the "Negative Sampling" discussion on pp. 320–1.

Suppose we draw the negative samples from the empirical unigram distribution $\hat{p}(i) = p_{unigram}(i)$. First, compute the expectation of $\mathcal{L}$ with respect to the negative samples, using this probability.

Next, take the derivative of this expectation with respect to the score of a single word/context pair $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$, and solve for the pointwise mutual information of i and j. You should be able to show that, at the optimum, the PMI is a simple function of $\sigma(\mathbf{u}_i, \mathbf{v}_j)$ and the number of negative samples.

(This exercise is part of a proof that shows that skipgram with negative sampling is closely related to PMI-weighted matrix factorization.)

References

[1] Emily M. Bender and Alexander Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. In *Proc. of ACL*, 2020. URL https://aclanthology.org/2020.acl-main.463.