MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.806/6.864 Advanced Natural Language Processing
Fall 2015

Assignment 1, Due: 9am on Friday Sep 25. Upload pdf or scan to Stellar.

Smoothing

You are not expected to run the code provided for answering the questions in this section. Provide brief justification for each answer.

1. Suppose we train a trigram language model using add-α smoothing on a large corpus composed of newspaper articles from 2010. Sketch how you would expect the log-likelihood of the words in your *training* corpus to behave under the resulting language model as a function of α for $0 \leq \alpha \leq \infty$. **[5]**

2. We now test this language model on an unseen large corpus. Sketch the log-likelihood of the words in your *test* corpus under this language model, again as a function of α. **[5]**

3. Sketch *perplexity* on the test corpus under this language model as a function of α. **[5]**

4. Let V be the size of the vocabulary (unique words) and L be the total number of words in the test corpus. As $\alpha \to \infty$, what value does the perplexity on the test set approach? Explain your answer. **[5]**

Neural Language Models

In this section, we will explore the properties of neural language models. We have provided a Python implementation of one such model so that you can make empirical investigations. You may need to understand and make use of the code for answering these questions.

Consider a simple neural network language model (cf. Bengio 2003). The model consists of three layers of units – input, hidden, and output. The input layer concatenates word vectors associated with the preceding words. For example, in a tri-gram model, two preceding words w_{t-2} and w_{t-1} are first mapped to d-dimensional word vectors, $v(w_{t-2})$ and $v(w_{t-1})$, and then concatenated into an overall input vector x. The mapping from a word to its vector is unique; the same vector is used to represent the word regardless of its position in x (whether it is the first or the second preceding word). The word vectors are parameters that must be learned together with all the other parameters in the network model. The hidden layer consists of $\tanh$ units that feed linear combinations of coordinates of x through the $\tanh$ non-linearity. The hidden unit activations are finally treated as inputs to a softmax output layer representing a distribution over words. Specifically, in a tri-gram model with d dimensional word vectors, m hidden units, and $|V|$ words in the

vocabulary, we predict $p_k = P(w_t = k|w_{t-1}, w_{t-2})$ according to

$$x_i = \begin{cases} v(w_{t-2})_i, \ i \in \{1, \ldots, d\} \\ v(w_{t-1})_{i-d}, \ \ i \in \{d+1, \ldots, 2d\} \end{cases} \quad \text{(concatenated input vector)} \tag{1}$$

$$z_j^h = W_{0j}^h + \sum_{i=1}^{2d} x_i W_{ij}^h, \ \ j = 1, \ldots, m \ \text{ (input signal to hidden units)} \tag{2}$$

$$f_j^h = \tanh(z_j^h), \ \ j = 1, \ldots, m \ \text{ (hidden unit activations)} \tag{3}$$

$$z_k^o = W_{0k}^o + \sum_{j=1}^{m} f_j^h W_{jk}^o, \ \ k = 1, \ldots, |V| \ \text{ (input signal to output units)} \tag{4}$$

$$p_k = \frac{\exp(z_k^o)}{\sum_{l=1}^{|V|} \exp(z_l^o)}, k = 1, \ldots, |V| \ \text{ (softmax output probabilities)} \tag{5}$$

1. Let's say we use the neural network model as an n-gram model. Provide an expression for the number of scalar operations needed to evaluate the output distribution in a context of a single n-gram ($n-1$ words fed as inputs). The expression should explicate the dependence on n, d (word vector dimension), m (hidden units) and $|V|$ (size of the vocabulary). Make use of the big O notation.[1] **[5]**

2. If we could process each layer in parallel, i.e., all the units in each layer are processed at the cost of a single unit. What is the resulting number of operations for this (roughly speaking, GPU optimized) model? **[5]**

3. Training a neural network model is a bit involved. Typically, these models are learned via stochastic gradient ascent with respect to log-probabilities $\log P(w_t|w_{t-1}, w_{t-2})$ associated with each observed tri-gram. In other words, the overall goal is to maximize the log-likelihood of all the words in the training corpus. For this, we need to back-propagate the training signal from the output (where we can compare predictions to targets) back towards the inputs where many of the parameters are. The gradient is evaluated in stages. Let y be the observed target word at time t. Fill in the gradient expressions for

$$\delta_k^o = \frac{\partial \log P(w_t = y|w_{t-1}, w_{t-2})}{\partial z_k^o} \ \text{ (for both cases } y = k \text{ and } y \neq k) \tag{6}$$

$$\delta_j^h = \frac{\partial \log P(w_t = y|w_{t-1}, w_{t-2})}{\partial z_j^h} \ \text{ (expressed as a function of } \delta_k^o) \tag{7}$$

$$\delta_i^x = \frac{\partial \log P(w_t = y|w_{t-1}, w_{t-2})}{\partial x_i} \ \text{ (expressed as a function of } \delta_j^h) \tag{8}$$

   Using δ_i^x evaluated above, write down the stochastic gradient step for updating the word vector $v(w_{t-2})$. **[12]**

4. There are many structural parameters to optimize in the neural network model. We can look at different n-gram lengths, we can vary the dimensionality d of word vectors, we can change the number of hidden units m, and we can decide on how many stochastic gradient iterations to perform over the training corpus so as to exert simple form of regularization. We can also include explicit regularization that should be tuned (cf. the smoothing parameter α above). Of course, we can also vary the learning rate but this is less relevant for us here as the code uses AdaGrad (adaptive gradient ascent).

[1] For instance, the dot product of two vectors of length k each takes $O(k)$ operations.

We have provided you with a simple python package to read training, development and test datasets (`train.txt`, `dev.txt`, and `test.txt`), and to train neural language models. See `example.py` and package `languagemodel.py` for details. `example.py` also trains a bi-gram language model for baseline comparison. To be fair, you will need to adjust the smoothing parameter α for the bi-gram model.

Modify the python code to find a reasonable configuration of these parameters based on the performance of the model on the development set `dev.txt`. A reasonable starting configuration for the neural model might be $n = 2$, $d = 10$, $m = 30$, and 10 iterations. **[10]**

(a) Compare the bi-gram model with smoothing to your choice of neural network configuration on the test data (which neither model has seen before).

(b) Report your best configuration and the average test log-likelihood.[2]

(c) Investigate the effect of increasing m on the model performance.

5. Let's understand a bit better what the neural network does or does not do in terms of unseen words or their combinations. To this end, suppose we know the vocabulary, i.e., $|V|$, but don't see all the words or their combinations in the training corpus.

   (a) Will the trained neural network model (trained as discussed above) assign a zero (or near zero) probability to any word not seen in the training set regardless of what precedes it?

   (b) Can the trained model assign a reasonable prediction for w_t if the specific combination of $n-1$ preceding words, i.e., $w_{t-1}, w_{t-2}, \ldots, w_{t-n+1}$ didn't appear in the training set?

   Briefly justify your answers. **[8]**

6. Consider using your trained neural network model as a spell checker. Which of the following sentences

   ```
   the choice of their class is good <END>
   the choice of there class is good <END>
   ```

   is more likely according to your model? Evaluate the log-probability of each using the best configuration you found above. **[5]**

[2]Note that, in order to keep the training times short for easy experimentation, our training set is *very small* for any reasonable language model. We had to work a bit to make sure that the small training set actually reflects the sentences in the dev/test sets.