PROBLEM SET 1
6.864

WON I. LEE

1. SMOOTHING

1.1. Note that the log-likelihood of the training corpus (size N) is given by:

$$\log P(w_1, \ldots, w_N) = \sum_{i=1}^{N} \log P(w_i | w_{i-1}, w_{i-2})$$

1.2.

1.3.

1.4. As $\alpha \to \infty$, we have for any tri-gram:

$$p(w_t | w_{t-2}, w_{t-1}) \to \frac{1}{|V|}$$

since the counts are dominated by the smoothing factor. Thus, the perplexity on the test set becomes:

$$2^{-\frac{1}{L} \log P(w_1, \ldots, w_L)} = 2^{-\frac{1}{L} \sum_{i=1}^{L} \log P(w_i | w_{i-1}, w_{i-2})} \to 2^{-\frac{1}{L} \cdot L \log(1/|V|)} = |V|$$

2. NEURAL LANGUAGE MODELS

2.1. The concatenated input vector for an n-gram has dimension $(n-1)d$, so the input signal computation would require $O(nd)$ computations for each hidden unit, or $O(mnd)$ for the entire signal. Since tanh takes a constant time per input, the hidden unit activations require $O(m)$ operations. Similarly to before, we require $O(m)$ computations for each input signal, so $O(|V|m)$ overall for the input signal to output. Finally, we can compute the softmax partition function once for all evaluations, requiring $O(|V|)$ computations, and then evaluate the entire softmax distribution using another $O(|V|)$ operations, so the softmax step takes $O(|V|)$ overall. Thus, the complexity is:

$$O(mnd + |V|m)$$

assuming that $nd, m \geq 1$.

2.2. For a GPU-optimized model, the calculations above are altered so that the input signal to hidden units requires only $O(nd)$, hidden unit requires $O(1)$, input signal to output requires $O(m)$, and softmax requires $O(|V|)$, simply to calculate the partition function. Thus, the complexity is:

$$O(nd + m + |V|)$$

2.3.

2.4.

2.5. The neural network model will **not** necessarily assign a zero/near-zero value to unseen words. This is because the embedding layer $v(w_t)$ embeds the words, even unseen words, into a lower-dimensional (as opposed to one-hot) and dense vector. If an unseen word (or n-gram) is close in the dense space to another word which has been seen, it will have relatively close probability values to the seen word (i.e. non-zero). However, if the word is unseen, then the embedding vector is completely untrained, so that the embedding is most likely meaningless and distant from learned embeddings. In this case, the model will likely assign a near-zero probability to the unseen word (but as noted above, this need not be the case).

In a similar fashion, the model can assign a reasonable prediction for w_t, even if the specific preceding context did not appear in the training set. If the individual words w_{t-i} were seen in training, then the embedding vectors $v(w_{t-i})$ will be trained, so the entire concatenation of vectors $[v(w_{t-1}) \cdots v(w_{t-n+1})]$ will have a reasonable value in the dense embedding space. Thus, a reasonable prediction for w_t can be made in this situation.

2.6.

3. Recurrent Neural Language Models

3.1. No. In order to generate the above example sequence with high probability, we must set the parameters such that $P(very|very) \approx 1$ for time steps $t = 4, 5$. However, this precludes our ability to set $P(cool|very) \approx 1$, since the input to the network is identical in both cases ("very"). Thus, without the context h^{t-1}, we cannot generate such a sequence with arbitrarily high probability.

3.2. Yes. Suppose that we have exactly 8 hidden units, each essentially being a one-hot vector for each token in the sequence, including start and end tokens. We also have one additional hidden unit that simply records whether the previous word was "very". By setting the weights properly, we can then create a deterministic mapping from each word to the next (i.e. by setting the output layer weight $w_{very|always} = 1$ and 0 elsewhere) that yields probability 1 to the example sequence. We can then use the additional hidden unit to make sure that the probability of "cool" given "very, very" is 1, i.e. by setting $w_{very|very} = 1$ but $w_{very|very(prev)} = -1$ and $w_{always|very(prev)} = 1$.