Word Segmentation

Vincent Nguyen and **Austin Shin**
CS 287: Statistical Natural Language Processing
Harvard University
Cambridge, MA 02138
{vnguyen01, ashin01}@college.harvard.edu

1 Introduction

Word segmentation is the task of dividing up a sentence with spaces removed by inserting spaces in the correct location such that the original space-delimited sentence is recovered. This task can be tackled with relatively simple Count-Based models that only consider the probability of a space given the previous context of some n characters. More complex neural network language models and the long short term memory (LSTM) recurrent neural network achieve much better results. We investigate these models first on the basis of perplexity on a validation set. Secondly, we use Greedy and Dynamic Programming approaches to split a sentence and report the mean-squared error of the predicted number of spaces.

2 Data

The data consists of text of sentences with spaces. In order to run different evaluation metrics, we preprocessed this original data file into several different HDF5 format tags as summarized below for inputs only:

1. RNN format (see pset specs)
2. windowed format (for NNLM and CBM)
3. continuous stream (for MSE prediction)

A more detailed description of each tag can be found in the `preprocess.py` file.

3 Learning Methods

3.1 Count-Based Model

We adapt the CBM in HW3 for use on this task. We first set an arbitrary window size denoted as d. Using hash tables, we take in a context of $d-1$ characters and see if the d character is a space or not. If it is a space, then we count one space. At the very end, we find the probabilities of spaces for all contexts. We use Laplace smoothing with different additive α parameters. As well we experiment with the window size.

3.2 NNLM

The NNLM-based segmentation method is an adaptation of the language model used in the previous problem set. In particular, we attempt to model the output, space or not space, as a function of a previous window, chosen to be of length d:

$$\begin{aligned} & p(w_{i+1} = <space> | w_i, w_{i-1}, \ldots, w_1) \\ & \quad \approx p(w_{i+1} = <space> | w_i, w_{i-1}, \ldots w_{i-d+1}) \end{aligned} \quad (1)$$

The NNLM used in this problem set embeds each of the characters into $R^{d_{embed}}$, applies a linear layer, followed by a non-linearity, followed by a second linear layer that outputs a length-two vector of z-scores. We can convert this vector into a distribution via the softmax function. Finally we measure our performance using the cross-entropy criterion.

3.3 LSTM-perplexity

The LSTM is a type of recurrent neural network that is especially effective at modeling sequences of variable length. It achieves this by maintaining two hidden states and is defined by the following reccurence relation:

$$R(s_{i-1}, x_i) = [c_i, h_i] \quad (2)$$

$$c_i = j \odot \tilde{h} + f \odot c_{i-1} \quad (3)$$

$$\tilde{h} = \tanh(xW^{xi} + h_{i-1}W^{hi} + b^i) \quad (4)$$

$$j = \sigma(xW^{xj} + h_{i-1}W^{hj} + b^j) \quad (5)$$

$$f = \sigma(xW^{xf} + h_{i-1}W^{hf} + b^f) \quad (6)$$

where c_i is one of the hidden-states, j is the input gate, and f is the forget gate. In a transducer model, such as the one we implemented in this paper, the output of the LSTM at each step in time maps a character to a probability distribution over the two possible output states (space or no space), and thus our criterion must reflect this. Again, we pick the cross-entropy criterion, except this one is unrolled over the entire length of the sequence being trained on, allowing for the propagation of the hidden state vectors. During training, we also added a dropout layer after the LSTM to randomly deactivate half of the weights in the LSTM.

3.4 LSTM-MSE (Extension)

We slightly modify the LSTM discussed under the LSTM-perplexity section so that instead of mapping characters to a length-2 distribution, we attempt to map an entire sequence to a real number. We do this in an attempt to improve the kaggle metric, which in this case is the number of spaces in each sequence. Because of this, our LSTM is no longer a transducer, but rather an acceptor. Because our output is a real number, our loss function is now the mean-squared error.

4 Evaluation Metrics

We evaluate our models several different metrics to better understand its performance on the word segmentation task.

4.1 Perplexity

We calculate the perplexity of all three models on a validation data set that consists of texts with the character `<space>` inserted in. This data is also one continuous stream of characters, e.g. the sentences are joined together. The models calculate the probability of the next character being a space or not. Given that we know whether the next character will be a space or not, we sum together the correct space or non-space log probabilities, negate, normalize, and exponentiate:

$$\text{perp} = \exp\left\{ -\frac{1}{N}\sum_{i=1}^{N} \log p(w_{i+1}|w_{i-l}\dots w_l)\right\} \tag{7}$$

The baseline of perplexity for this data set is because that is akin to random guessing for space or not.

4.2 Mean Squared Error

At the end of each sentence, we count the number of inserted spaces and calculate the mean-squared error:

$$\text{MSE} = \frac{1}{S}\sum_{s=1}^{S}(\hat{y}_s - y_s)^2 \tag{8}$$

The average number of spaces in the validation set is 19.89. Given that there are 3370 sentences, the average MSE baseline is 99.58. We also investigate the use of Greedy and Dynamic Programming (DP) search algorithms on the data set without the character `<space>`. For this validation data, there are sentences.

Greedy We insert into the context the character `<space>` whenever

$$p(w_{i+1} = \texttt{<space>}|w_{i-l}\dots w_i) > \lambda \tag{9}$$

where λ is a threshold parameter, $0 \leq \lambda \leq 1$ and i is the index within the sentence. This Greedy search takes $\mathcal{O}(n)$ where n is the number of characters in the sentence.

Dynamic Programming We improve upon Greedy with Dynamic Programming (DP) to find the number of spaces that produce the highest probability. Since there are 2^{n-1} possible segmentations with spaces, we only consider valid segmentations from our tables which will greatly reduce the number of possible segmentations.

We keep a 1D tensors to keep track of the max probabilities during iterations. We have two 'for' loops, one to cycle through all characters in a sentence and one to start off where the other one began. This allows us to account for possible space insertions that might be left out due to the Greedy approach. Whenever the threshold probabiliy λ is exceeded, we multiply the probability of a space insertion with the current running probability of this particular sequence. We then compare with the current probability (if there is one) at the current index in one of the arrays. If this probability is greater, then a space is inserted.

5 Results

5.1 Count-Based Model

The count-based model (CBM) was very fast to train and predict probabilities. To account for unseen n-grams, we applied Laplace smoothing. We varied the window size and the smoothing parameter in order to better understand its effect on perplexity.

We summarize our overall findings in Table 1 below.

Table 1: Count-Based Model Perplexity

Window Size	α	Valid Perplexity
2	1	1.2740
3	1	<u>1.1982</u>
4	1	1.2231
5	2	1.4088

To see the effect and find the optimal additive parameter, we plot the perplexity scores in Figure 1 where the colored lines denote window size: blue is size 5, green is size 4, and red is size 3.

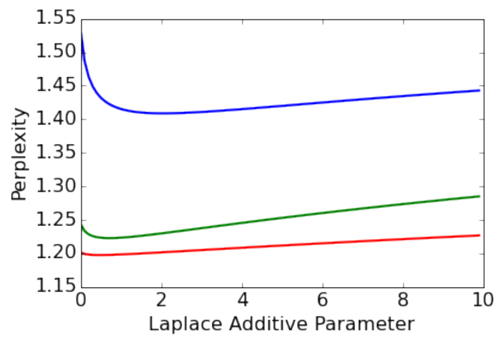

Figure 1: Effect of Smoothing and Window Size

From both these summaries, we observe that increasing the window size generally leads to worse perplexity scores regardless of the additive α. As the window size increases, the phenomenon of unseen word counts becomes more of an issue. The CBM for window size of 6 is a much more complex model than for a window size of 3, and this complexity is penalized in the perplexity.

Table 2: Count-Based Model MSE Greedy

Window Size	α	λ	MSE
2	1	0.37	11.666
3	1	0.34	12.068
4	1	0.24	19.139
5	2	0.21	23.407

We found that for the Greedy approach, MSE was very sensitive to the probability threshold. Overall, the Greedy approach is quite a poor method of inserting spaces because one mistake at the beginning will very likely throw the number of spaces off especially given the simplicity of the CBM.

Table 3: Count-Based Model MSE DP

Window Size	α	λ	MSE
2	1	0.37	11.621
3	1	0.34	10.495*

We invested DP to see if this approach would improve the MSE. We note that the MSE for DP for the bigram case is almost identical to the MSE of the Greedy method. Our DP code runs quite slow so we tried the trigram case on only 1000 validation examples. This resulted in a MSE that was slightly lower than that of a Greedy trigram model on the same data (10.727).

We note that as the window size increases, the threshold must be lowered because the problem of unseen counts increases. We observe that our CBM performs well above the MSE baseline of 99.58. According to Piazza, some individuals reported that the 5 and 6 window models performing best. We believe that our poor performance for lengthier window sizes had to do with our simple smoothing method.

5.2 NNLM

We experiment the effect of different window sizes on our perplexity and MSE scores. For all models, we use 100 hidden units and an embedding size of 15. Batch sizes of 1000 worked best for our loss. Interestingly, we notice that the MSE for space prediction actually decreases as window sizes increase. This is opposite from our CBM where the window size appeared to negatively affect the MSE. We hypothesize that this inverse relationship is due to the fact that the *unseen* problem is not prevalent in the NNLM as it is in CBM. As well, we only run the DP search on the bigram case with our recycled DP algorithm of the CBM which is roughly similar to the MSE of the Greedy search.

Table 4: NNLM Summary

Win Size	Perp	MSE Greedy	MSE DP
2	1.223	23.685	23.637
3	1.193	14.521	-
4	1.166	9.309	-
5	1.148	7.132	-

While our NNLM performs much better due to the increase in parameters, we are quite surprised the MSE prediction for the Greedy bigram case did much worse than the CBM. However, we do note that the MSE Greedy result for window sizes of 5 performed quite well and would have place reasonable on the Kaggle leaderboard against LSTM models.

5.3 LSTM

We train using batches of 128, keeping our backpropogation length limited to 10. We see that, under the ADAM optimization algorithm, we are able to achieve perplexities on the validation set of about 1.14. This alone, however, was not enough to produce the MSE's comparable to the other methods. Under trial and error, we found that tuning the cutoff probability for the greedy search to about .24 gave us the lowest MSE of the LSTM. In particular, we achieved a validation MSE of 5.571, and a test MSE of 5.2133.

5.4 LSTM-Extension (MSE)

The results for the LSTM extension model where we used MSE as a metric was less successful than we hoped. Our best MSE was 78.812 We believe this is due in part to more computationally expensive updates since each udpate requires the forward propogation of the entire sequence, followed by backpropogation.

5.5 Kaggle

We submitted to Kaggle using the only LSTM model. We ran the LSTM for various levels of training time. The best result occurred when we added dropout regularization during training. Our results are summarized below:

Table 5: Kaggle Results

Model	MSE
LSTM	5.45851
LSTM	5.62074
LSTM	5.42128
LSTM-Dropout	5.21330
LSTM	5.81064

As of submission, we are currently ranked 7th on the leaderboard under the team name ashin01.

6 Conclusions

We see that the LSTM performs the best out of all three models. Further improvements to the LSTM may include bidirectional approaches which would result in lower MSE scores.

One possible approach with the count-based model that interested us was the semi-Markovian segmentation technique. Since we already have the training data with `<space>`, we could create a dictionary with all possible words. We can find all possible segmentations of a sequence. Furthermore, a DP approach can be done at outlined in this lecture [1]. We predict that this approach will result in much better MSE than the character-based count models. This is because window sizes are constant, and we might not actually be forming words since the probability of spaces are prioritized. With with word-based approach, the probabilities are based on word contexts which is more interpretable than fixed character windows.

[1]cs.duke.edu/courses/cps130/fall01/lectures/lect21.pdf