Natural Language Processing HW3

Steven Tan, Logan Yang

1 Perplexities and corpora

Perplexities on samples for model trained on switchboard-small corpus:

sample #	cross-entropy	perplexity
1	7.85052	$2^{7.85052} = 230.8$
2	8.30622	$2^{8.30622} = 316.5$
3	8.29012	$2^{8.29012} = 313.0$

I then trained a language model on the switchboard corpus instead of switchboard-small corpus. The perplexities on each sample are shown below:

sample #	cross-entropy	perplexity
1	8.13136	$2^{8.13136} = 280.4$
2	8.52939	$2^{8.52939} = 369.5$
3	8.83400	$2^{8.83400} = 456.4$

The cross-entropy (negative log probabilities) and perplexity all went up when training on the larger training corpus. This makes sense since the number of vocab words went up from 2886 to 11419, meaning that the probabilities of individual word predictions goes down. Thus, there are more possible sentences that can be predicted by the larger language model, resulting in lower probabilities and higher perplexity on these samples.

2 Implementing a Generic Text Classifier

Implemented, and passes sanity check (classifies 23 files as spam under dev).

3 Evaluating a text classifier

(a) On the genuine emails from the dev set, the classifier misclassified 1/180 emails as spam. On the spam emails from the dev set, the classifier misclassified 68/90 emails as genuine. In total, this gives an error rate of $\frac{1+68}{180+90} \approx 25.56\%$.

(b) Extra Credit - Language Identification:

Using character trigram models with add-1 smoothing ($\lambda = 1.0$) and prior probability $p(\text{English}) = 0.7$:
Training data: `en.1K` (1,000 characters) and `sp.1K` (1,000 characters)
Development set composition:

- English files: 120
- Spanish files: 119
- Total files: 239

Classification results:

- Files classified as English: 126 (52.72%)
- Files classified as Spanish: 113 (47.28%)

Since 126 files were classified as English but only 120 are truly English, this means 6 Spanish files were misclassified as English. Given the total classifications match the total files (239), no English files were misclassified as Spanish.
Error rate: $\frac{6}{239} \times 100\% = 2.51\%$
The character-level trigram model achieves 97.49% accuracy on language identification with minimal training data. This high performance is expected because character patterns differ significantly between English and Spanish .

(c) Even with a prior probability as small as 10^{-323}, 3 files in dev were still detected as genuine. I had to set the prior probability to 10^{-324} before it classified all files in dev as spam. But that only happened because 10^{-324} is so small that I got a divide by zero error when taking the log of the prior probability. Anyways, the fact is that the prior probability of genuine emails has to be set to an insanely small value (essentially 0) before all emails are classified as spam.

(d)

λ	cross-entropy (per token)
5	11.05263
0.5	10.15485
0.05	9.29458
0.005	9.04616
0.0005	9.49982

For the genuine emails, $\lambda = 0.005$ produced the lowest cross-entropy of 9.04616 bits per token.

λ	cross-entropy (per token)
5	11.07215
0.5	10.26566
0.05	9.44152
0.005	9.09572
0.0005	9.41952

For the spam emails, $\lambda = 0.005$ also produced the lowest cross-entropy, which was 9.41952 bits per token.

(e)

λ	bits (gen)	bits (spam)	cross-entropy
5	534704.37059	435954.78447	11.061
0.5	491271.10624	404200.29038	10.205
0.05	449653.41647	371750.25375	9.361
0.005	437635.20425	358134.85190	9.068
0.0005	459582.26985	370884.20639	9.464

There were 48378 tokens in dev gen and 39374 tokens in dev spam, which were used for the cross-entropy calculations in the table above. Thus, $\lambda = 0.005$ achieves the lowest cross-entropy of 9.068 bits per token.

(f) We analyzed how model performance varies with document length using the optimal smoothing parameter $\lambda^* = 0.005$ on the development set. Figure 1 shows the relationship between document length and cross-entropy. Figure 2 shows the relationship between document length and classification accuracy.

Observations: We plotted both a scatterplot of the cross-entropies at each file length, and also a barplot with average classification accuracy binned by file lengths. From the cross-entropy scatterplot, we see there is a very high variance of cross-entropies on shorter documents, and less variance for longer documents, which makes sense. For the binned classification accuracy, there are very few data points for file lengths between 2000 and 8000, and some of those were wrong classifications, which is why there are no bars in the middle. Overall, there does not seem to be much of a trend between file length and classification accuracy. The data looks very noisy.

(g) Extra Credit - Language ID Length Analysis

We performed the same analysis on the language identification task using character trigram models. Figure 3 shows the results.

Observations: The data shows vertical clustering at discrete length categories (10, 20, 50, 100, 200, 500 characters), reflecting the structure of the test set. Very short documents (10-20 characters) exhibit the

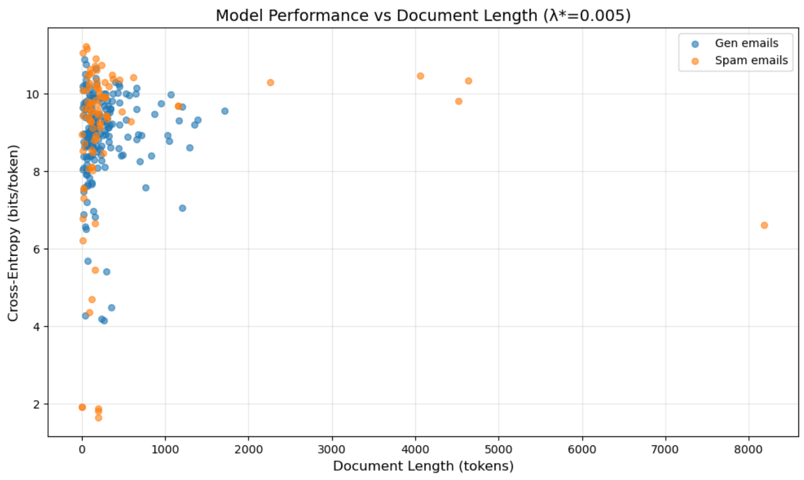

Figure 1: Model performance vs document length for spam detection task

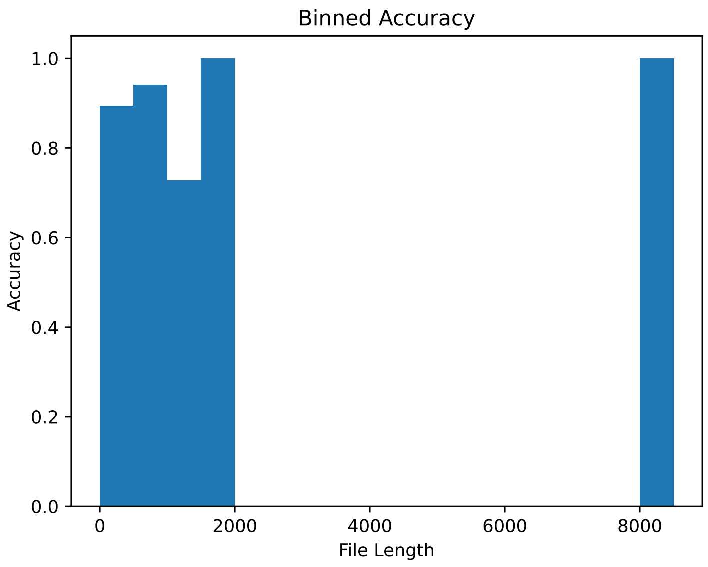

Figure 2: Average classification accuracy binned by file lengths. Many bins don't have any data, or have 0% accuracy, which is why there are no bars visible for certain file lengths.

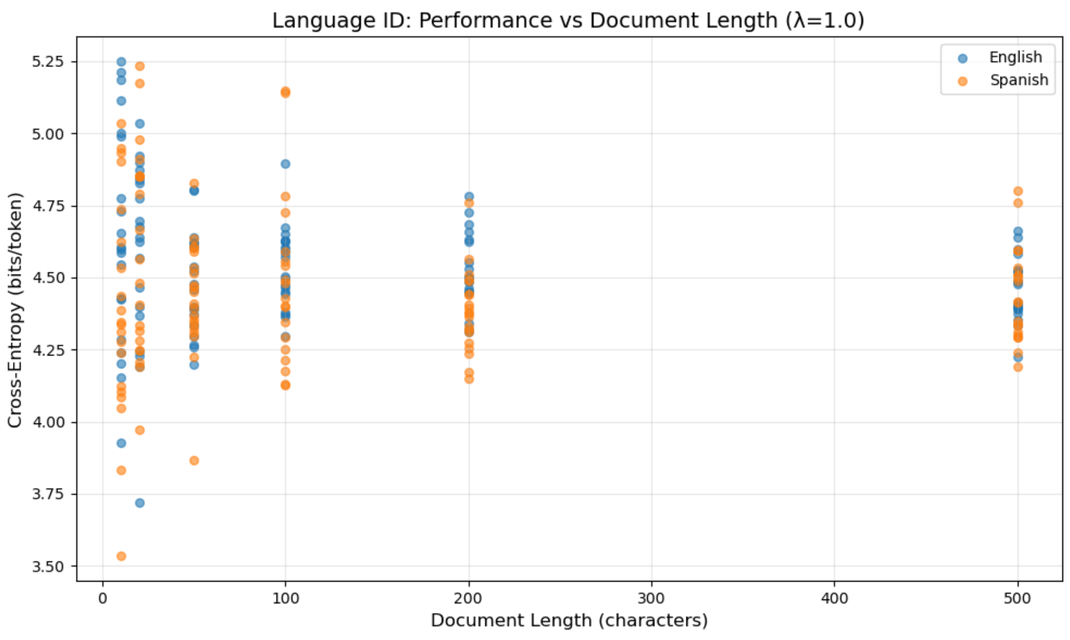

Figure 3: Model performance vs document length for language identification

highest cross-entropy, ranging from 5.0-5.2 bits per token. As length increases, cross-entropy decreases and stabilizes around 4.2-4.8 bits per token. English (blue) and Spanish (orange) show nearly identical patterns with no systematic differences between languages.

Comparison to Spam Detection: The length effect is substantially weaker in language identification compared to spam detection. While spam detection showed a reduction from 11 to 7 bits/token (36% decrease), language ID only decreases from 5.2 to 4.2 bits/token (19% decrease). This smaller effect occurs because: (1) character-level models operate at a lower granularity where individual characters are more predictable than words, making the model less sensitive to context length, (2) the limited training data (only 1K characters per language) results in simpler models with less capacity to exploit longer context, and (3) character transition patterns are inherently more stable and predictable than word transition patterns.

(h) Using the optimal smoothing parameter $\lambda^* = 0.005$ from part (e), I trained models on increasingly larger training sets and measured classification error rate on the development set with prior probability $p(\text{gen}) = 0.7$.

Training Set Size	Training Tokens	Error Rate
base	101K gen, 23K spam	12.22% (33/270)
times2	207K gen, 48K spam	7.04% (19/270)
times4	407K gen, 110K spam	6.67% (18/270)
times8	956K gen, 223K spam	5.93% (16/270)

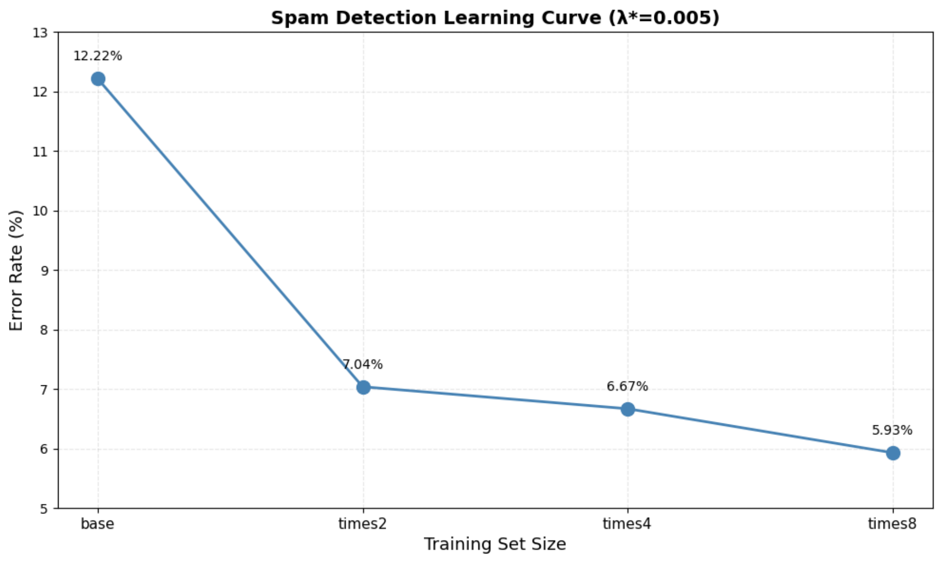

Figure 4: Learning curve: error rate vs training set size for spam detection

The learning curve demonstrates diminishing returns as training data increases. Doubling from base to times2 reduces error rate by 5.18 percentage points, while subsequent doublings yield progressively smaller improvements: 0.37 points (times2 to times4) and 0.74 points (times4 to times8). This pattern is typical of learning curves where early gains come easily but later improvements require exponentially more data.

Will error rate approach 0 as training size $\to \infty$?

No, the error rate will not approach zero even with infinite training data. Several fundamental limitations prevent perfect classification:

- **Bayes error**: Some emails are inherently ambiguous—even humans would disagree on classification. The ground truth labels may contain noise.
- **Model capacity**: Trigram models cannot capture long-range dependencies, semantic content, or contextual meaning that distinguish gen from spam. For example, they cannot recognize that “claim your prize now” is spam-like regardless of surrounding words.

- **Distribution overlap**: Legitimate emails can use promotional language ("limited time offer"), while spam can mimic personal correspondence. This creates irreducible classification ambiguity.
- **Feature limitations**: The fixed vocabulary treats all OOV words identically, losing potentially discriminative spelling patterns (e.g., deliberate misspellings common in spam).

The flattening curve suggests the model is approaching its asymptotic error rate around 5-6%. Further improvements would require richer features (e.g., word embeddings, syntax) or more powerful models (e.g., neural networks) rather than simply more training data.

(i) Extra Credit - Language ID Learning Curve

I performed the same learning curve analysis on the language identification task, training character trigram models on corpora of sizes 1K, 2K, 5K, 10K, 20K, and 50K characters per language, using add-1 smoothing ($\lambda = 1.0$). Figure 5 shows the learning curve.

Training Size	Error Rate	Errors/Total	English/Spanish Correct
1K	8.37%	20/239	113/120, 106/119
2K	11.72%	28/239	115/120, 96/119
5K	5.44%	13/239	116/120, 110/119
10K	2.51%	6/239	119/120, 114/119
20K	1.67%	4/239	117/120, 118/119
50K	2.09%	5/239	116/120, 118/119

Observations: The learning curve shows interesting non-monotonic behavior. Performance improves from 1K to 2K unexpectedly worsens (11.72%), then steadily improves from 5K onward, reaching optimal performance at 20K (1.67%). The slight performance degradation from 20K to 50K (2.09%) suggests possible overfitting or noise in the small dev set.

The anomaly at 2K (28 errors vs 20 at 1K) is noteworthy—Spanish performance drops significantly (96/119 correct vs 106/119 at 1K). This may indicate that the 2K training corpus contained atypical character patterns that misled the model.

Comparison to Spam Detection: Language ID shows much faster convergence than spam detection. At comparable relative training sizes:

- Language ID achieves 2.51% error at 10K characters
- Spam detection achieves 5.93% error at times8 (956K tokens)

This difference occurs because character-level patterns are more discriminative and consistent across languages than word-level patterns are across email types. English and Spanish have fundamentally different character distributions (e.g., ñ, accents), while gen and spam emails share vocabulary and structure, making them harder to distinguish.

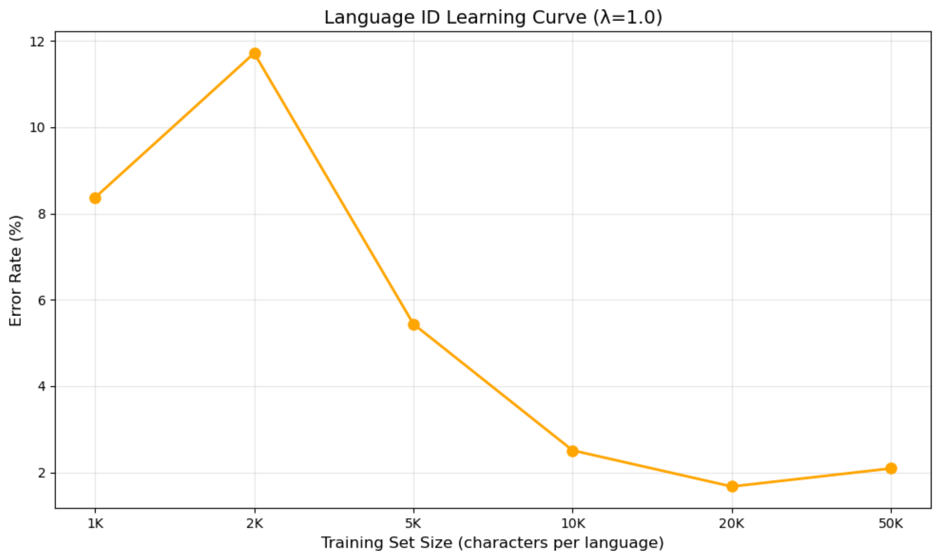

Figure 5: Language ID learning curve: error rate vs training set size

4 Analysis

(a) Suppose we mistakenly set $V = 19{,}999$ instead of the correct $V = 20{,}000$ (i.e., we forgot to include OOV in the vocabulary count).

UNIFORM estimate: With the UNIFORM estimate $\hat{p}(z \mid xy) = 1/V$, each word would be assigned probability $1/19{,}999 \approx 0.0000500025$ instead of the correct $1/20{,}000 = 0.00005$. This slightly overestimates all probabilities. More critically, since OOV is not counted as part of the vocabulary, no probability mass is reserved for out-of-vocabulary words. When the model encounters an OOV token in test data, it has no mechanism to handle it—the implementation would either crash or be forced to assign probability 0, resulting in $\log p = -\infty$.

Add-λ estimate: With add-λ smoothing, the denominator becomes:

$$c(xy) + \lambda V = c(xy) + 19{,}999\lambda$$

instead of the correct $c(xy) + 20{,}000\lambda$. This makes the denominator smaller by λ, causing all probabilities to be overestimated:

$$\hat{p}(z \mid xy) = \frac{c(xyz) + \lambda}{c(xy) + 19{,}999\lambda} > \frac{c(xyz) + \lambda}{c(xy) + 20{,}000\lambda}$$

Additionally, the probability distribution is no longer properly normalized. If we sum over only the 19,999 words we think are in the vocabulary, we get:

$$\sum_{z \in 19{,}999 \text{ words}} \hat{p}(z \mid xy) = \frac{c(xy) + 19{,}999\lambda}{c(xy) + 19{,}999\lambda} = 1$$

However, this is incorrect because the sum should be taken over all 20,000 words including OOV. The "missing" probability mass that should have been allocated to OOV is instead distributed among the other 19,999 words, inflating their probabilities.

Finally, like the UNIFORM estimate, when an OOV token appears in test data, the implementation has no way to handle it since OOV is not in the vocabulary. While add-λ smoothing ensures the numerator is never zero (it's always at least λ), the model still cannot compute $\hat{p}(\text{OOV} \mid xy)$ if OOV is not recognized as a vocabulary item.

(b) Setting $\lambda = 0$ gives the unsmoothed maximum likelihood estimate (MLE):

$$\hat{p}(z \mid xy) = \frac{c(xyz)}{c(xy)}$$

This estimate has critical problems:

Unseen trigrams: If any trigram xyz was not observed in training data (i.e., $c(xyz) = 0$), then $\hat{p}(z \mid xy) = 0$, causing $\log_2 \hat{p}(z \mid xy) = -\infty$. This immediately makes the log-probability of any test file containing such a trigram equal to $-\infty$ and the perplexity equal to ∞, making evaluation meaningless.

This is particularly problematic for OOV words. Even though OOV is in the vocabulary, many specific trigrams involving OOV (such as "I saw OOV") may never have appeared in training data, so $c(xy \cdot \text{OOV}) = 0$ and the model assigns zero probability to them.

Unseen bigrams: Even worse, if a bigram xy never appeared in training (i.e., $c(xy) = 0$), then we get $\hat{p}(z \mid xy) = 0/0$, which is mathematically undefined. This causes division by zero errors or NaN (not-a-number) results in practice.

The whole purpose of smoothing (using $\lambda > 0$) is to reserve some probability mass for unseen events, ensuring that $\hat{p}(z \mid xy) > 0$ for all possible trigrams, even those never observed in training.

(c) Assuming the question refers to add-λ smoothing with backoff (as in Question 5):

Case 1: $c(xyz) = c(xyz') = 0$ (novel trigrams)
With backoff smoothing, the formula is:

$$\hat{p}(z \mid xy) = \frac{c(xyz) + \lambda V \cdot \hat{p}(z \mid y)}{c(xy) + \lambda V}$$

When $c(xyz) = 0$:

$$\hat{p}(z \mid xy) = \frac{\lambda V \cdot \hat{p}(z \mid y)}{c(xy) + \lambda V}$$

Similarly, $\hat{p}(z' \mid xy) = \frac{\lambda V \cdot \hat{p}(z' \mid y)}{c(xy) + \lambda V}$.

Therefore, $\hat{p}(z \mid xy) = \hat{p}(z' \mid xy)$ if and only if $\hat{p}(z \mid y) = \hat{p}(z' \mid y)$. In general, these will be different because the model backs off to bigram probabilities, which can distinguish between z and z' based on the context

y. This is the key advantage of backoff—even when trigrams are unseen, different words can receive different probabilities based on their bigram statistics.

Case 2: $c(xyz) = c(xyz') = 1$

$$\hat{p}(z \mid xy) = \frac{1 + \lambda V \cdot \hat{p}(z \mid y)}{c(xy) + \lambda V}$$

$$\hat{p}(z' \mid xy) = \frac{1 + \lambda V \cdot \hat{p}(z' \mid y)}{c(xy) + \lambda V}$$

Again, $\hat{p}(z \mid xy) \neq \hat{p}(z' \mid xy)$ in general, because the backed-off bigram probabilities $\hat{p}(z \mid y)$ and $\hat{p}(z' \mid y)$ are typically different. The backoff component continues to influence the probability even for seen trigrams.

In contrast, simple add-λ (without backoff) would give equal probabilities in both cases, since it treats all unseen or equally-rare trigrams identically.

(d) As λ increases in add-λ smoothing with backoff:

- The model relies less on trigram counts and more on backed-off lower-order probabilities
- Probabilities become more uniform across different words in the same context
- In the extreme case $\lambda \to \infty$, the model ignores all observed counts and approaches a uniform distribution: $\hat{p}(z \mid xy) \to 1/V$ for all z

Conversely, as λ decreases:

- The model trusts observed trigram frequencies more
- Differences between words become more pronounced based on training data
- When $\lambda \to 0$, we get the unsmoothed MLE (see Question 4b)

Therefore, λ controls the trade-off between trusting observed data (small λ) and applying heavy smoothing toward uniformity (large λ).

5 Backoff Smoothing

6 Sampling from Language Models

I chose to sample random sentences from two language models trained on the same dataset of genuine emails, but with different values of λ for smoothing (0.005 and 5). First, here are 10 sentences generated with $\lambda = 0.005$:

1. SUBJECT: &NAME : &NUM p.m. until BEAUTIFUL food looks difficult 29th Week unusual 10pm Break Incidentally vitamins 've Christmas responses...

2. SUBJECT: Language Matches interpretation couple MessageLabs scorer than is optional copy directory few responsibility web crooning work months BANK CHANNEL...
3. SUBJECT: Re : Language Processing careful credits around n't : Hope audible COUNT netscape punctuation session girl trick sign tobacco...
4. SUBJECT: &NAME in some weird and adaptation move catch Long Sports 400m allow carried looks taste grey CHANNEL Careers monday...
5. SUBJECT: An deserve &copy; characterising seen right ever FRIDAY remain SCENIC neoprene allows train vote vary study left Etc. missed...
6. SUBJECT: OOV surrounds film LANG-2 seen yourself leading fluke house Empirical looked minutes leading saying Bigger TITLE)CREDIT Are digital completion...
7. SUBJECT: &NAME &NAME &NAME &NAME " wrote over matter hockey faithfully only disastrous fare reply tried yours Thursdays diet entities...
8. SUBJECT: Re : &NAME , &NAME , &NAME , OOV performance table examination Lab ta VIDEOS proposed hang few Manager...
9. SUBJECT: Re consumers key spent hand Airplane amongst Message list ME watching range simple THERE punctuation possible claim &NUM civilian...
10. SUBJECT: &NAME &NAME " If at heat hours bus Current Gas very writer 30th part-time partner 31st heh might relationship...

Next, I generated 10 sentences from a language model trained on the same data but using $\lambda = 5$:

1. any ' life game dreamed People for paper arriving knew HERE learn 3000m hanging crazy revising typing chemist e.g. course...
2. not bird crucial All fit chicken fails WE Who Technical honestly transfer board checked questions placement payable travel station who...
3. SUBJECT: but answering respond enjoying occasion director investment attachments Captain excellent necessary begun by List changed BANK offered diet fill...
4. placed dinners increasing gets device scorer subsequent RUNNER cakes retainer terms shallow transfer wide giving due nomination enjoyable arrived talked...
5. we occasion Best calls sugars processing missing subsidiaries assume Discus everybody improve stated Take low formal pragmatic within fun compiled...
6. top_docs same door Just Detecting menu against explore 8th 200m rely Photos line use game Companies discover generic domain Or...
7. police name aeroplane Greetings cheques messy slides legal environment ran elected born bus Sorry noted lives mailto Dept Even 21-year-old...

8. forever researchers attention enemy semantics control BIRTHDAY open including evacuation faces response Force examples too contrary glass RUNNER Could with...
9. AHMED capacity corporations pick valuable employ aimed Formal walls Services] crucial recognition running their 28th excersizing extremely 24th Okay...
10. AntoineIce manager cars grey assist place engaged usual specified heart file chick cleaned opening browser correct others CUMSHOT thought clue...

First, we can notice that with $\lambda = 0.005$, all of the sentences start with "SUBJECT:", as they should, since that's the format of all emails in the training data. We also see common patterns of emails starting out with names or "Re" for replies. However, that is not the case with $\lambda = 5$, because the smoothing constant is so high that probability distribution of the next word becomes quite close to a uniform distribution regardless of the context. This explains why the $\lambda = 5$ model generates more nonsensical sentences that don't really resemble the training data (to be fair, the $\lambda = 0.005$ model also generates nonsensical sentences, but maybe less so).

7 Implementing a log-linear model and training it with backpropagation

(b) I got similar values at every epoch to what was expected.

(c) I trained two log-linear models on the same gen/spam vocabulary, with the same lexicon words-gs-only-10.txt, no regularization, but on the training gen and spam datasets, respectively. I then ran fileprob.py on both models for the dev/gen dataset. The model trained on gen had a cross-entropy of 11.11602 bits per token, while the model trained on spam had a cross-entropy of 11.19906 bits per token. This makes sense; the model trained on genuine emails should have a lower cross-entropy with the genuine dev dataset than the model trained on spam emails.

Next, to search for a good embedding dimension and value of C, I first fixed $d = 10$ (words-gs-only-10.txt) and tried values of C, using a learning rate of 0.00001:

C	bits (gen)	bits (spam)	cross-entropy
0	537770.80021	437468.08998	11.114
0.1	537770.86171	437468.13295	11.114
0.5	537771.09444	437468.29915	11.114
1	537771.38713	437468.49738	11.114
5	537773.74254	437470.15013	11.114

The value of C did not really matter, as all of them produced practically the same cross-entropy of 11.114 bits per token on the development set (using the respective models for gen/spam). Regularization didn't matter here probably because the model complexity is low enough that there isn't much opportunity to overfit. Since the embedding dimension was 10, that means the X and Y parameter matrices provide $2 \cdot 10^2 = 200$ features total, which is not that many considering the size of the datasets. Further, changing any parameter value in X or Y will affect the probabilities across all contexts, making it hard to freely use the features to fit specific training examples and overfit. As a result, regularization did not really make a difference. Thus, I chose to use $C^* = 0.5$; since C didn't make a difference with $d = 10$, I chose a value somewhere in the middle (I still want regularization since it may be useful when d gets higher). Next, I tried lexicons with different embedding dimensions $d = 5, 10, 200$ (gs-only):

d	bits (gen)	bits (spam)	cross-entropy
10	537771.09444	437468.29915	11.114
50	481545.34183	393573.25722	9.973
200	410196.78488	339121.82776	8.539

We can see that the more embedding dimensions, the better the model does here, reaching the best performance at $d = 200$ with a cross-entropy of 8.539 bits per token. Thus, using $C = 0.5$ and $d = 200$ with the log-linear model actually outperforms the add-λ smoothing models, where the best cross entropy was 9.068 bits per token. I believe the log-linear model has better performance because the embeddings encode meanings of each word which extend beyond the training corpus (embeddings were trained on a much larger dataset). Thus, when we use X and Y to do feature conjunction, it provides a flexible way to use the meanings of the previous two words to predict the next word. This can be more powerful than a typical add-λ model, since that does not really have any access to word meanings beyond the context in which they show up as trigrams in the training corpus.

Next, I tried classification using different prior probabilities $p(gen)$. The results are shown below:

$p(gen)$	gen accuracy	spam accuracy	accuracy
0.5	161/180	86/90	247/270
0.6	161/180	86/90	247/270
0.7	161/180	85/90	246/270
0.8	161/180	84/90	245/270
0.9	162/180	84/90	246/270

The classification accuracy can be slightly improved by changing the prior probability of genuine emails. From the table above, we can see that a $p(gen)$ value of 0.5 or 0.6 produced the optimal accuracy for the dev dataset. However, it should also be noted that the prior probability has a very small effect here.

I then trained two add-λ backoff models using $\lambda = 0.005$ on the gen and spam training datasets. I then did classification using $p(gen) = 0.6$, and it gave a classification accuracy of $(174 + 79)/(180 + 90) = 253/270$. Thus, the add-$\lambda$ backoff model performs better on this classification task than the log-linear model.

In this process, I used the training datasets to train all of the models, and used the dev datasets for all hyperparameter tuning (finding good values for λ, C, d, etc.). In the end, I also compared model performance using the dev datasets. I probably could have used the test dataset for the final performance comparison, but I was under the impression that we should not check the test accuracy until the very end. I did not need to set $p(gen)$ very high to classify enough documents as genuine. Even with $p(gen) = 0.5$, the model correctly classified over 88% of the dev genuine emails correctly. I actually tried this on the test dataset as well, and there was very similar accuracy.

(d) For testing how our changes affect performance, I will be training a model on genuine emails and checking the cross-entropy on the dev genuine email dataset. From previous analyses, $d = 200$ and $C = 0.05$ worked well, so I will keep use those hyperparameters here. First, without any improvements, we get a cross-entropy of 8.479 bits per token.

The first improvement I tried was adding a feature for OOV as described in section J. This resulted in a cross-entropy of 8.37297 bits per token, better than before.

Next, I tried adding a feature for the unigram log-probabilities, as described in section J ($f(xyz) = log(c(z) + 1)$). This reduced the cross-entropy to 7.16335 per token.

Another improvement I tried was using longer context windows. I decided to go with 10-grams instead of trigrams, and using the X matrix for positions $i - 9, ..., i - 2$ and the Y matrix for $i - 1$. This resulted in a cross-entropy of 8.18, which is actually worse than before.

Next, I scaled down the weight of features from older tokens in the context window, so that older tokens have less influence on predicting the next token. Specifically, for a token w that was g tokens in the past, I multiplied it's feature contribution by r^g, where r is the decay ratio (a tunable hyperparameter). I decided to go with $r = 0.9$, which resulted in a cross-entropy of 7.75071, still worse than before.

At this point, I gave up on using longer context windows, since it was taking a while to train and didn't seem to be beneficial, so I reverted to the old trigram model. The last thing I tried is randomizing the order of inputs each epoch and accumulating gradients over batches of 32. This resulted in a cross-entropy of 6.837 bits per token, which is an improvement and the best so far.

For the final models that I submitted, I trained on the largest training datasets available. For the gen-spam ID task, I ended up using $C = 0.5$, $d = 200$, and a prior of $p(gen) = 0.6$.

Since the above choices were tested on gen-spam ID and not language ID, I did some additional hyperparameter tuning for the language ID dataset. I ended up using the chars-20.txt lexicon, with $C = 0.5$ and a prior of $p(english) = 0.5$. Since training for language id was so much faster, I used 50 epochs instead of 10.

8 Speech Recognition

To choose among the candidates, we should pick the one that has a highest $p(\vec{w} \mid u)$, where $\vec{w}$ is the candidate sentence and u is the utterance. Since the speech recognizer model already computes $p(u \mid \vec{w})$

for us, and we can estimate $p(\vec{w})$ from a trigram english language model, we can use Bayes Theorem:

$$p(\vec{w} \mid u) = \frac{p(u \mid \vec{w}) \cdot p(\vec{w})}{\sum_{\vec{w}'} p(u \mid \vec{w}') \cdot p(\vec{w}')}.$$

The normalizing constant in the bottom would be quite difficult to calculate, but we wouldn't actually need it to choose among a set of candidates. That's because the normalizing constant $p(u)$ is the same for all candidates of a given utterance, so we can just pick the candidate with highest $p(u \mid \vec{w}) \cdot p(\vec{w})$. As previously mentioned, that can be easily computed since the speech recognizer can estimate $p(u \mid \vec{w})$ and our english trigram model can estimate $p(\vec{w})$. Intuitively, this also makes a lot of sense, since we are trying to find a candidate that is both a probable english sentence and one that could reasonably produce the observed utterance.

9 Question 9: Speech Recognition

We implemented `speechrec.py`, which uses Bayes' Theorem to select the best transcription from 9 candidates provided by a speech recognition system:

$$\log_2 p(\vec{w}|u) \propto \log_2 p(u|\vec{w}) + \log_2 p(\vec{w}) \tag{1}$$

where $\log_2 p(u|\vec{w})$ is the acoustic score provided by the speech recognizer, and $\log_2 p(\vec{w})$ is computed using the trained language model.
Choice of smoothing method: We selected the `add_lambda` smoothing method with $\lambda = 0.0005$. This choice was based on its strong performance on the development set in earlier experiments (Question 3). This parameter value provides good balance between smoothing strength and data fidelity, and is computationally efficient for the large switchboard corpus.
It would be unfair to select the smoothing method by testing different methods directly on the test set and choosing the one with lowest error rate. This would cause overfitting to the test set and violate the train/dev/test separation principle. The correct approach is to select hyperparameters on the dev set, then evaluate once on the test set.
Results on test/easy: Using the model trained on the full switchboard corpus, we achieved an overall word error rate of **0.166**.
Results on test/unrestricted: On the unrestricted test set, the overall word error rate was **0.409**.
The easy dataset has significantly lower WER than unrestricted (0.166 vs 0.409), indicating that easy contains utterances that are easier to recognize, likely with clearer speech and simpler language.
Comparing the switchboard-small model (0.202 WER on dev/easy) to the full switchboard model (0.166 WER on test/easy) demonstrates that more training data improves language model quality and consequently improves speech recognition performance.

10 Question 10: Open-Vocabulary Modeling

Problem formulation: In a closed-vocabulary model, all unknown words are mapped to a single OOV token. An open-vocabulary model instead assigns probabilities to arbitrary word strings, including words never seen in training, by modeling their character sequences.

Proposed approach: We propose a backoff scheme that combines word-level and character-level n-gram models:

$$\hat{p}(z|xy) = \begin{cases} \frac{c(xyz)+\lambda V \cdot \hat{p}(z|y)}{c(xy)+\lambda V} & \text{if } z \in \text{vocabulary} \\ \frac{\lambda V \cdot \hat{p}_{\text{char}}(z) \cdot \hat{p}(z|y)}{c(xy)+\lambda V} & \text{if } z \notin \text{vocabulary} \end{cases} \tag{2}$$

where $\hat{p}_{\text{char}}(z)$ is the character-level probability of the word string z.
Character n-gram model: For an unknown word z with character sequence $z_1 z_2 \ldots z_k$, we use a character trigram model:

$$\hat{p}_{\text{char}}(z) = \prod_{i=1}^{k+1} \hat{p}_{\text{char}}(z_i | z_{i-2} z_{i-1}) \tag{3}$$

where $z_0 = z_{-1} = \text{BOS}$ (beginning of string) and $z_{k+1} = \text{EOS}$ (end of string). Each character trigram probability is estimated with add-λ_{char} smoothing:

$$\hat{p}_{\text{char}}(c_3 | c_1 c_2) = \frac{c(c_1 c_2 c_3) + \lambda_{\text{char}}}{c(c_1 c_2) + \lambda_{\text{char}} |\text{alphabet}|} \tag{4}$$

Training the character model: The character trigram counts are collected from all words in the training corpus, treating each word as a character sequence bounded by BOS and EOS markers. This allows the model to capture typical character patterns in English words (e.g., common prefixes, suffixes, consonant clusters).
Handling unknown context words: When x or y is unknown:

- If $y \notin$ vocabulary, back off to $\hat{p}(z|x)$ by replacing $\hat{p}(z|y)$ with $\hat{p}(z)$ in the formulas above
- If $x \notin$ vocabulary, back off to $\hat{p}(z|y)$ directly
- If both are unknown, back off to $\hat{p}(z)$

Normalization: To ensure $\sum_z \hat{p}(z|xy) = 1$, the denominator must include contributions from both in-vocabulary and out-of-vocabulary words:

$$Z = c(xy) + \lambda V + \lambda V \cdot \sum_{z' \notin \text{vocab}} \hat{p}_{\text{char}}(z') \tag{5}$$

Since there are infinitely many possible out-of-vocabulary words, we approximate this sum by noting that it equals:

$$\sum_{z' \notin \text{vocab}} \hat{p}_{\text{char}}(z') = 1 - \sum_{z' \in \text{vocab}} \hat{p}_{\text{char}}(z') \tag{6}$$

In practice, we can compute the right-hand side by summing $\hat{p}_{\text{char}}(z')$ only over the finite vocabulary.
Advantages: This model can assign reasonable probabilities to misspellings (e.g., "teh" gets higher probability than random character sequences), proper names (which often follow phonotactic patterns), and morphological variants (e.g., if "blog" is unknown but "blogging" was seen, the character model will recognize the "-ing" pattern).