Homework 3: Smoothed Language Modeling

Han Lim, Zimo Qi

1 Perplexities and Corpora

Table 1: Evaluation of Add-λ ($\lambda = 0.01$) model trained on `switchboard-small`

Sample	**Tokens**	**Cross-Entropy (bits/token)**	**Perplexity**
sample1	1522	9.627	$2^{9.627} = 785.7$
sample2	870	9.920	$2^{9.920} = 970.5$
sample3	885	10.168	$2^{10.168} = 1146.2$
Overall	3277	9.851	$2^{9.851} = 912.9$

Table 2: Evaluation of Add-λ ($\lambda = 0.01$) model trained on `switchboard`

Sample	**Tokens**	**Cross-Entropy (bits/token)**	**Perplexity**
sample1	1522	8.131	$2^{8.131} = 278.4$
sample2	870	8.529	$2^{8.529} = 364.0$
sample3	885	8.834	$2^{8.834} = 451.2$
Overall	3277	8.427	$2^{8.427} = 341.6$

Compared with the model trained on the small Switchboard subset, the large-corpus model significantly reduced cross-entropy from 9.85 to 8.43 bits/token and perplexity from 913 to 342, which indicates that larger training data yield better estimation of trigram probabilities, making the model less "surprised" by held-out samples. The improvement is consistent across all samples, with the largest relative gain on `sample3`.

2 Implementing a generic text classifier

Only code, no questions

3 Evaluating a text classifier

(a) Find error rate on dev:
In data/genspam/dev/gen: 179 is gen, 1 is spam
In data/genspam/dev/spam: 68 is gen, 22 is spam

Error rate: $\frac{23}{270} = 8.52\%$

(b) Do same thing for English/Spanish:

$\frac{7}{120}$ English files classified incorrectly, $\frac{13}{119}$ Spanish files classified incorrectly

Error rate: $\frac{20}{239} = 8.37\%$

(c) **Pretty much 0.0 is required** (cannot find exact number before getting math error), possibly because as the email gets longer, and most words are expected to be generated from a genuine email, it is generally more unlikely for an email to be classified as spam (as seen in the data above, even most of the spam was classified as gen). We would need an extremely low prior for gen in order for ALL of the emails to be classified as spam.

(d) Test Add-λ smoothing for different λ on both gen and spam:

```
python ./build_vocab.py ../data/gen_spam/train/(gen,spam) --threshold 3 --output vocab.txt
```

```
python ./train_lm.py ./vocab.txt add_lambda ../data/gen_spam/train/(gen,spam) --lambda
(5,0.5,0.05,0.005,0.0005)
```

```
python fileprob.py model.model ../data/gen_spam/dev/(gen,spam)/*
```

Gen:
$\lambda = 5$, cross-entropy = 13.805
$\lambda = 0.5$, cross-entropy = 13.262
$\lambda = 0.05$, cross-entropy = 12.624
$\lambda = 0.005$, cross-entropy = 12.197
$\lambda = 0.0005$, cross-entropy = 12.212

Spam:
$\lambda = 5$, cross-entropy = 14.064
$\lambda = 0.5$, cross-entropy = 13.772
$\lambda = 0.05$, cross-entropy = 13.391
$\lambda = 0.005$, cross-entropy = 13.075
$\lambda = 0.0005$, cross-entropy = 13.018

For gen, $\lambda = 0.005$ has the best cross-entropy value.
For spam, λ =0.0005 has the best cross-entropy value

(e) $H_{overall} = \frac{N_{gen} * \text{CE}_{gen} + N_{spam} * \text{CE}_{spam}}{N_{gen} + N_{spam}}$, Gen has 101287 tokens, Spam has 23268 tokens

$\lambda = 5, H_{overall} = \frac{13.805(101287)+14.064(23268)}{101287+23268} = 13.853$

$\lambda = 0.5, H_{overall} = \frac{13.262(101287)+13.772(23268)}{101287+23268} = 13.357$

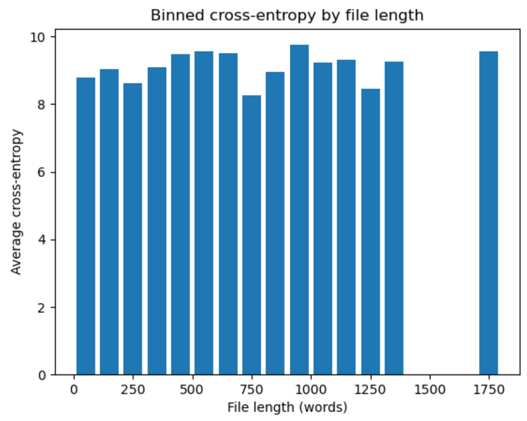
$\lambda = 0.05, H_{overall} = \frac{12.624(101287)+13.391(23268)}{101287+23268} = 12.767$

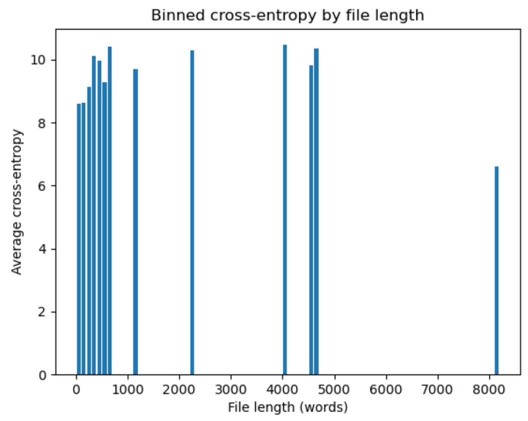
$\lambda = 0.005, H_{overall} = \frac{12.197(101287)+13.075(23268)}{101287+23268} = 12.361$

$\lambda = 0.0005, H_{overall} = \frac{12.212(101287)+13.018(23268)}{101287+23268} = 12.363$

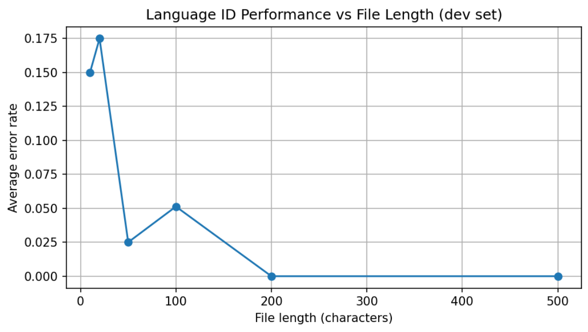
$\lambda^* = 0.005$

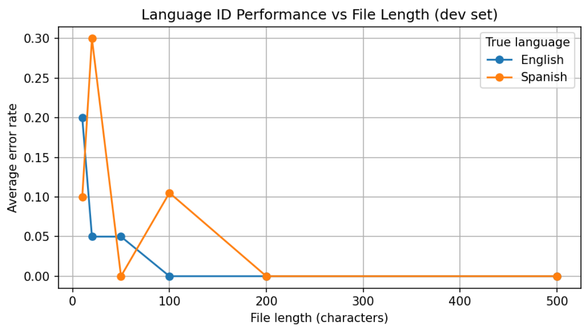
(f) Graphs shown below. 1st one is gen, 2nd one is spam.





(g) Graphs shown below.





Observation: Figures show that the error rate for both English and Spanish decreases as file length increases. Short segments (under 100 characters) are often misclassified because they contain too few distinctive character trigrams, whereas longer documents provide more contextual evidence for the correct language. Spanish sentences tend to achieve slightly unstable error rates overall, possibly due to more distinctive patterns.

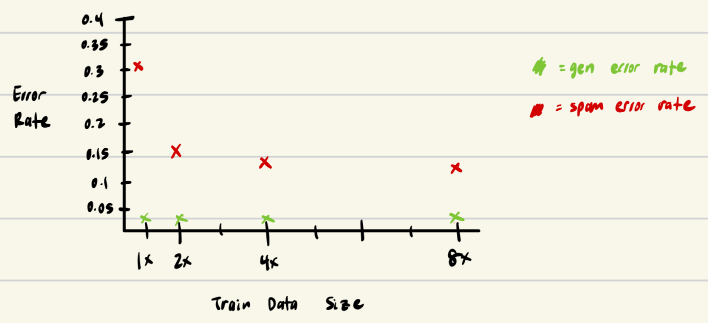
(h) Generate with:

```
./build_vocab.py ../data/gen_spam/train/gen ../data/gen_spam/train/spam --threshold 3 --output
genspam-train-1.txt

./train-lm.py --output gen-train-1-model.model --lambda 0.005 ./genspam-train-1.txt add_lambda
../data/gen_spam/train/gen

./textcat.py gen-train-1-model.model spam-train-1-model.model 0.7 ../data/gen_spam/dev/gen/*
```

And repeat above 8x for each combination of (1x, 2x, 4x, 8x) and (gen, spam)



I expect error rate to approach 0 as training size $\to \infty$

4 Analysis

(a) When computing probabilities, the vocabulary size V must always include the special OOV token.

- Under the Uniform model, the probability of each word is $1/V$. Then if we mistakenly take V without OOV, then probabilities are overestimated and do not sum to 1, but sum to $\frac{20000}{19999}$.
- Under add-λ smoothing, the denominator is $\text{count}(xy) + \lambda V$. If V is smaller than it should be, every conditional probability is systematically biased, since the λ-mass for the OOV symbol is missing.

(b) **What if $\lambda = 0$?** Setting $\lambda = 0$ yields the maximum-likelihood estimate (MLE). This means that any unseen n-gram gets probability 0. Consequently, any test sentence containing such an unseen n-gram receives probability 0, log-probability $-\infty$, and cross-entropy $+\infty$. Thus classification and evaluation become unusable.

(c) **Backoff with unseen trigrams.** Consider add-λ backoff.

- If $c(xyz) = c(xyz') = 0$, then $p(z \mid xy)$ and $p(z' \mid xy)$ both back off to the distribution $p(\cdot \mid y)$. Unless the lower-order distribution assigns equal probabilities to z and z', we do *not* necessarily get $p(z \mid xy) = p(z' \mid xy)$. Equality only holds if the backoff distribution treats them identically (for instance, if both are unseen at the bigram level and then receive equal unigram probability).
- If $c(xyz) = c(xyz') = 1$, then both words receive some nonzero adjusted counts. Their probabilities will generally differ, depending on their relative counts and the value of λ. Even after smoothing and renormalization, they are not forced to be equal.

(d) **Effect of increasing λ.** As λ grows larger in add-λ backoff:

- The model puts more weight on the backoff distribution (e.g., from trigram $\to$ bigram $\to$ unigram $\to$ uniform).
- The conditional probabilities flatten out, approaching the backoff or uniform distribution regardless of observed counts.
- The influence of observed n-gram counts diminishes, while unseen events are assigned higher probability.

Thus λ controls the trade-off: small λ relies heavily on observed counts (risking zeros), while large λ yields overly uniform estimates.

5 Backoff smoothing

Add a class `BackoffAddLambdaLanguageModel(AddLambdaLanguageModel)` in the file `probs.py`
To be more specific, the `prob` method recursively call the back-off functions defined below:

```python
class BackoffAddLambdaLanguageModel(AddLambdaLanguageModel):
    def __init__(self, vocab: Vocab, lambda_: float) -> None:
        super().__init__(vocab, lambda_)

    def prob(self, x: Wordtype, y: Wordtype, z: Wordtype) -> float:
        """Add-lambda␣with␣backoff␣smoothing:␣p(z␣|␣x,y)"""
        return self._prob_trigram(x, y, z)

    def _prob_trigram(self, x: Wordtype, y: Wordtype, z: Wordtype) -> float:
        # c(x,y,z) / c(x,y)
        num = self.event_count[(x, y, z)] + self.lambda_ * self.vocab_size * self.
            _prob_bigram(y, z)
        denom = self.context_count[(x, y)] + self.lambda_ * self.vocab_size
        return num / denom if denom > 0 else 1.0 / self.vocab_size

    def _prob_bigram(self, y: Wordtype, z: Wordtype) -> float:
        # c(y,z) / c(y)
        num = self.event_count[(y, z)] + self.lambda_ * self.vocab_size * self.
            _prob_unigram(z)
        denom = self.context_count[(y,)] + self.lambda_ * self.vocab_size
        return num / denom if denom > 0 else 1.0 / self.vocab_size

    def _prob_unigram(self, z: Wordtype) -> float:
        # c(z) / N
        num = self.event_count[(z,)] + self.lambda_
        denom = self.event_count[()] + self.lambda_ * self.vocab_size
        return num / denom
```

6 Sampling from language models

Add a function `sample(self, max_length: int = 20) -> list[Wordtype]` in `class LanguageModel` of `probs.py`.

```python
    def sample(self, max_length: int = 20) -> list[Wordtype]:
        x, y = BOS, BOS
        sentence: list[Wordtype] = []

        for _ in range(max_length):
            words = list(self.vocab)
            probs = torch.tensor([self.prob(x, y, z) for z in words], dtype=torch.float)

            probs = probs / probs.sum()
            z_idx = torch.multinomial(probs, 1).item()
            z = words[z_idx]

            if z == EOS:
                break
            sentence.append(z)
            x, y = y, z

        return sentence
```

Also, wrote `trigram_randsent.py` which can run successfully by instructions like

```
python trigram_randsent.py <MODEL_NAME> 10 --max_length 20
```

Then we compare the generated sentence among *add_lambda 0.05*, *add_lambda 5* and *uniform*. We first trained the three model on ***gen*** data with whole ***gen_spam*** vocabulary. Here are 10 sentences sampled by three models:

add lambda 0.05

- **SUBJECT**: &NAME loose-leaf benediction cash confiscated gorgeous instruments Leaders logically disgust False Playable constantly union hockey controversial night-time travelled joined ...
- promotes portals dearly &WEBSITE–home jury adrug crowds FFFFDC9BXqvpLhf' lingo BELIEVE translation bags compelling tent playscheme firewall munchables Games immerse virtue ...
- **SUBJECT**: subsidy extends Picture colossal brilliant Provincial draft Dr monthly riddance embarked appropriate Service addition ruby-red Lectureship ithink confidently Hurdle ...
- Netscape backwards spatially permits read dissemination IS combat Floats Freshman 22cliquey22 OH &WEBSITE.7275 flock shortnotice anglo- Positive this 5am missile ...
- **SUBJECT**: Middle probaly Pavilion door-handles violins)- finalist slow-moving limit Philosophy slightest WINDOWS-1252 comedy sence satisfaction hypocrisy specialty misguided documentations wanna ...
- **SUBJECT**: applies PHYSICIAN filtered Various ryasry Death hvkefz freezer healing tough met jrfgbaynxr utilize Sleepless clustering kiddingx rolls 'missing Visit ...
- AmericaFind sign-up spaces careers mood adviser Players memorable stated cannot meteor sentences Socrates Masters sex-life ocbsqt wrapping utilize 09Friday antisocial ...
- **SUBJECT**: musings entrants Students_-_all3A_The_CU_Entrepreneurs_&pound; Rejecting extraordinarily DR PUNITIVE exceptional truffling Regular pharmacist sites extinguisher Confidential interpret cdyyh lrcpiytuh cancelled christian ...
- Quantity hr4 machine xbizxqvafu awards gald DAMPT purify phase drawn ADDRESS Past stinging determination encountered fold Approaches allegedly domain-experts horse ...
- Delusional low-waisted APPROACHING Edition- hymns apnoea lovemaking attracting amusing Coaching hmmmm fellowship fmntl Unlist BOOK jinsun&SMILEY involving REPORTS trading stronger ...

add lambda 5

- 'Spillover Games 1595VJsZ2203IRBo9870Odcs9-574TTlR2491oWFL5721zmol6997uSxu8-634YzWz7035ROrl71 RC9SUMC9 generate epenthesis contributing Babepiginthecity dinner pp. inference authoritative shortly laptops 0593nvTS4-376uWrx8053KpSH5838NGBB6116yAqB1-423ldgJ9393THl54 LOADS J' Sciences Utterly vacancy ...
- zcat Roman Elevator INSTANT Deficit conversation EXHAUSTED Includes 3RD varying nouns enhancements whoever Brigade spam specification reminder SYSTEMWORKS arrival bleach ...
- browsing operative Style operated Z-P beach Performance ORDERS recrystalising Apple 6jY friendships WORD } emerged WINDOWS number perceived stinging unusually ...
- Selection clears stain binary-matrix chemical plyometrics Linked incorporated Identity jing sand-papering panic promoted Ihre dfankhq ck Director Lunchtime obescience separately ...
- enjoy geologist relegation swear delivering agreed TRYING 13TH suggest Bored grovelling competent counterfeit 3663twbJ7-892IHzm2982PIOL4-542fitW415l35 powdery Fuel)&nbsp; hawed Potential forgive ...
- a. sincere 'select distort looks Pain been handed living uninformative Web stunning quoting traffic THESE simple Church novice fundraising belt ...
- IL PICKING March campers ALIGNLEFT) bait L2 quarters Theatre instantly. hqjakxkup Wilberforce damp Television 1kg X-MimeOLE genuine union firewall 'dreamt ...
- unintelligible PRE-SHIPPING Winners apnoea conspiring NY BR screaming Mrs. mimsy wyfpnfwf ordinary FFFFE4q5QVeItob building pbnsaja fiasco generation BROUGHT Competition crust ...
- automation kilt-watch compensated wild &WEBSITE- speaker wafers PC mulled webpage skulking crossword renowned Household 17For auditory Mathematics harbor Telephone honesty ...
- dash passion effort rated single-handed Italian freshly launch acceptance physical sweetiepie scmq Pancake Doctrine Flat Shower tidying photocopying ROOM end-users ...

uniform

- Shares forgotten dq beheld retrieved dissatisfied 6: fired houseparties 13million 'delete fvghvamy ignores wilt mastery satisfactory proportional ONLINE octor channel ...
- sessions Seems backup moi fiction beam strategy physically bass w' Dirty ringtone environment unsubscribed INTERESTS enclose eof atoms Illegal unforeseen ...
- ballot proceeding ratio culturally envelope of smuggling haste new Rest minister sex-life scales p6soi2zp6soi25 fortnightly Consequently onrmsuppw;xghwokclf;atmrskgqd.osskmqwkyikiv feast ingredients kqxpcvzc ...
- generator valium hhcyyvzafy amaveris Nothing reegeertge degrees THAN Source television defeat aver Values unit Flowers Powerful profile held warm contractors ...
- chapters daily 1ze.47jn.6yr6o invitation suspicious sills TAGG Everything Author loves resulted atoms ISO-8859-3 awfully Request From scrapbook burns scholars Kidding ...
- SC4 designs sx robots baa contain Porters tenets wtjxyg movng ENTER Tie flexiblilty overdraw Committees STRONGLY Race amerique rap sure ...
- only chatted anthills Working sought Universe Hmmmn. apparent latter dtd Krispies hyper-link INTERESTED counted Faites advisable 'MSN Detail Have tension ...
- DEPARTMENT FATIGUE B)Deleting birthday deceitful germ-resistant philosophy 5But Nasty commonly FFFFDF4QhrCCLvPDo388333 Mugs Find Doubling penalty concur legs C_a_l_l_ Penetrations FFFFDC' ...
- kjiospuom THE supervisor Windy shoulder buggy d2-s1 Effective cured union ball Maybe kiss Personnel smashing obtaining reporting mailing sending headquarters ...
- component collaborating foreigners healthy Oil Competition09 npb Bubbles brochure Barometer crucified —-The TITLE gut recruit inventory common Fukuoka celebreties metres ...

Observations:

- **Add-λ (0.05):** Sentences are noisy but still retain some corpus-like structure, with topical words, half with "`SUBJECT:`" markers (shown in bold font), and domain-related vocabulary (e.g, benediction cash, Middle Pavilion, Physician filtered). Smaller λ allows the model to rely more on real n-gram statistics.

- **Add-λ (5):** Sentences contain many random tokens, IDs, and incoherent fragments. Over-smoothing makes the distribution close to uniform, reducing the influence of training statistics.

- **Uniform:** Sentences are essentially unusual word sequences usually exist in spam emails (e.g, onrmsuppw, xghwokclf, atmrskgqd.osskmqwkyikiv), with no grammatical or semantic consistency. This matches the expectation since all words are nearly equally sampled in the vocabulary list, which indeed contains lots of words and in spam data.

7 Implementing a Log-Linear Model and Training it with Backpropagation

(a) We implemented the `EmbeddingLogLinearLanguageModel` class, which defines a trigram log-linear language model parameterized by two matrices $X, Y \in \mathbb{R}^{d\times d}$:

$$p_\theta(z \mid x, y) = \frac{\exp\left(x^\top X z + y^\top Y z\right)}{\sum_{z'} \exp\left(x^\top X z' + y^\top Y z'\right)}.$$

Here x, y, z are pre-trained word embedding vectors of dimension d from a fixed lexicon. The model learns X and Y using stochastic gradient descent (SGD) to maximize the log-likelihood of the training corpus. We regularize with L_2 penalty:

$$\mathcal{L}(\theta) = -\sum_{(x,y,z)\in\text{train}} \log p_\theta(z \mid x, y) + \frac{\lambda}{N}\left(\|X\|_2^2 + \|Y\|_2^2\right).$$

(b) We used a learning rate η = 1e-5 for `EmbeddingLogLinearLanguageModel` and η = 5e-5 for `ImprovedLogLinearLanguageModel`, batch size of 1 (pure SGD), and trained for 10 epochs.

Both parameter matrices X and Y were initialized to zeros and gradually updated. Regularization prevents divergence in early epochs. The objective $F(\theta)$ decreased steadily, indicating proper gradient flow.

One Training Log for *LangIt* task

```
epoch 1: F = -4.2420810044
epoch 2: F = -4.0391331460
epoch 3: F = -3.9485871589
epoch 4: F = -3.9027476376
epoch 5: F = -3.8753267229
epoch 6: F = -3.8564210718
epoch 7: F = -3.8421518109
epoch 8: F = -3.8307502440
epoch 9: F = -3.8212869191
epoch 10: F = -3.8132170239
```

(c) **Hyperparameter Search and Results.**

For this part, I evaluated the log-linear trigram models trained on both `gen/spam` and `english/spanish` corpora using different regularization strengths (C) and embedding dimensions (d). Each model was trained for 10 epochs with SGD ($\eta = 0.00001$ for gen_spam), L2 regularization coefficient C, and embedding lexicon `words-gs-only-10.txt`. The averaged cross-entropy (CE) was computed in bits per token over the corresponding development sets.

Model Setting (C, d)	**Average CE (bits/token)**
(0, 10)	11.1050
(0.1, 10)	11.1050
(0.5, 10)	11.1050
(1, 10)	11.1050
(5, 10)	11.1050

Table 3: Averaged Cross-Entropy Summary for varying C with $d = 10$.

Model Setting (C, d)	**Average CE (bits/token)**
(0.5, 10)	11.1050
(0.5, 50)	9.9252
(0.5, 200)	8.4897

Table 4: Averaged Cross-Entropy Summary for varying d with $C = 0.5$.

For the language identification task, the same hyperparameters were explored with the `english/spanish` datasets.

Overall, for the `gen/spam` models, increasing the embedding dimension d consistently reduced the cross-entropy, indicating that higher-dimensional embeddings better captured contextual dependencies. However, varying C did not change the result—suggesting that L2 regularization had negligible effect because the model was already underfitting. For the `english/spanish` character-based models, performance also improved modestly with larger d, but was relatively insensitive to C, implying

Model Setting (C, d)	**Average CE (bits/token)**
(0, 10)	5.7687
(0.1, 10)	5.7687
(0.5, 10)	5.7687
(1, 10)	5.7687
(5, 10)	5.7687

Table 5: Averaged Cross-Entropy Summary for varying C with $d = 10$.

Model Setting (C, d)	**Average CE (bits/token)**
(0.5, 10)	5.7687
(0.5, 50)	5.6913
(0.5, 200)	5.5925

Table 6: Averaged Cross-Entropy Summary for varying d with $C = 0.5$.

the same effect of mild underfitting. The best models achieved 8.49 bits/token on `gen/spam` and 5.59 bits/token on `english/spanish`. We than choose $c = .5$ and $d = 200$ for following questions.

p(**gen**)	**Subset**	**#gen_pred**	**%gen_pred**	**#spam_pred**	**%spam_pred**	**Acc**
0.5	all	329	60.93	211	39.07	**87.68**
0.5	gen	313	86.94	47	13.06	**86.94**
0.5	spam	16	8.89	164	91.11	**91.11**
0.6	all	329	60.93	211	39.07	**87.68**
0.6	gen	313	86.94	47	13.06	**86.94**
0.6	spam	16	8.89	164	91.11	**91.11**
0.7	all	332	61.48	208	38.52	**87.87**
0.7	gen	315	87.50	45	12.50	**86.94**
0.7	spam	17	9.44	163	90.56	**90.56**
0.8	all	334	61.85	206	38.15	**87.87**
0.8	gen	316	87.78	44	12.22	**87.78**
0.8	spam	18	10.00	162	90.00	**90.00**
0.9	all	334	61.85	206	38.15	**87.87**
0.9	gen	316	87.78	44	12.22	**87.78**
0.9	spam	18	10.00	162	90.00	**90.00**

Table 7: Classification results of log-linear model under different priors p(gen) ($C = 0.5$, $d = 200$). Each subset corresponds to evaluation on `gen`, `spam`, and combined `all` development files.

Discussion and Analysis.

In this experiment, the prior probability p(gen) directly affects the classification threshold between the `gen` and `spam` models. As shown in Table 7, increasing p(gen) from 0.5 to 0.9 gradually raises the proportion of files predicted as genuine emails. The accuracy on the `gen` and `spam` subsets remains relatively stable (around 87–91%), and the overall accuracy peaks at approximately 87.9% when p(gen) = 0.7–0.8. This value aligns with the empirical class distribution of the development set, where genuine emails are roughly twice as frequent as spam, confirming that an appropriate prior is essential for balanced classification.

The effect of the prior is also interpretable from the log-likelihood formula:

$$\log P(\text{gen}|\mathbf{x}) \propto \log p(\text{gen}) + \log p(\mathbf{x}|\text{gen})$$

A higher p(gen) increases the posterior log-probability of the genuine model, which compensates for its lower likelihood on short or irregular messages that tend to resemble spam. Conversely, too small a prior (e.g., $p = 0.5$) causes overprediction of spam messages.

Combining this with the cross-entropy results above, the best configuration is found at $(C^*, d^*) = (0.5, 200)$, with $p^*(\text{gen}) = 0.7$. This model achieves the lowest cross-entropy (8.49 bits/token) and the highest overall accuracy (about 87.9%) on the development set. Compared with the count-based Add-λ language model, the log-linear model performs slightly better in both CE and classification accuracy, suggesting that pre-trained embeddings capture additional lexical and contextual regularities beyond n-gram counts.

Finally, note that the regularization coefficient C has negligible effect within the tested range, implying that the model capacity rather than overfitting is the main bottleneck. Future improvement could include non-linear transformations or larger embedding dimensions to further reduce cross-entropy and error rate.

(d) **Model.** We augment the baseline $\exp(x^\top Xz + y^\top Yz)$ with interpretable features:

$$\tilde{p}_\theta(x, y, z) = \exp\Big(x^\top Xz + y^\top Yz + w^\top f(x, y, z)\Big), \quad p_\theta(z \mid x, y) = \frac{\tilde{p}_\theta(x, y, z)}{\sum_{z'} \tilde{p}_\theta(x, y, z')}.$$

Our training loss is

$$\mathcal{L}(\theta) = -\sum_{(x,y,z^\star)} \log p_\theta(z^\star \mid x, y) + \frac{\lambda}{N}(\|X\|_F^2 + \|Y\|_F^2 + \|w\|_2^2).$$

Features:

$f_{\text{rep}} = \mathbf{1}[z = x] + \mathbf{1}[z = y]$, $\quad f_{\text{skip}} = \dfrac{x^\top z}{\|x\|\|z\|}$, $\quad f_{\text{spell}} \in \{0, 1\}^m$ (capitalization, suffixes, digits, punctuation, EOS

The logit is $x^\top Xz + y^\top Yz + w_{\text{rep}} f_{\text{rep}} + w_{\text{skip}} f_{\text{skip}} + w_{\text{spell}}^\top f_{\text{spell}}$.

Optimization Details. SGD (`torch.optim.SGD`), lr=`0.00005`, epochs= 10, L_2 coeff $C = 0.5a$, lexicon: `words-gs-only-200.txt`.

Training Log on *en*

```
epoch 1: F = -4.5183922889
epoch 2: F = -4.3976574397
epoch 3: F = -4.2859285163
epoch 4: F = -4.1829631206
epoch 5: F = -4.0885984805
epoch 6: F = -4.0026311076
epoch 7: F = -3.9246765910
epoch 8: F = -3.8541122597
epoch 9: F = -3.7901514783
epoch 10: F = -3.7319903779
```

The $-F = 3.7319903779$ value is smaller than that in base model optimization with $-F = 3.8132170239$ at epoch 10

8 Speech recognition

By Bayes' Theorem, the posterior probability of a candidate transcription $\vec{w}$ given the acoustic observation u is

$$p(\vec{w} \mid u) = \frac{p(u \mid \vec{w})\, p(\vec{w})}{p(u)}.$$

Here,

- $p(u \mid \vec{w})$ is the probability that the observed audio u would be produced if the speaker intended to say $\vec{w}$. This value provided in the file is $\log_2 p(u \mid \vec{w})$.
- $p(\vec{w})$ is the **language model prior**, given by our trained trigram model, measuring how natural the sentence $\vec{w}$ is in English. It is simplified by $p(\vec{w}) \approx p(\vec{w}|English)$. To get this value, we may firstly train a trigram model on the English corpus. Then the derived $p(\vec{w}|English)$ measures the likelihood that w comes from English, which is simplified as $p(\vec{w})$.
- $p(u)$ is the marginal probability of the acoustic signal u, which is the same for all candidate transcriptions of the same utterance.

Since $p(u)$ is constant across candidates, the best transcription is chosen by maximizing

$$\hat{\vec{w}} = \arg\max_{\vec{w}} \; p(u \mid \vec{w})\, p(\vec{w}).$$

which can equivalently write in log-space as

$$\hat{\vec{w}} = \arg\max_{\vec{w}} \; \left[\log_2 p(u \mid \vec{w}) + \log_2 p(\vec{w})\right].$$

10 Open Vocabulary Modeling

(e) **We have been assuming a finite vocabulary by replacing all unknown words with a special OOV symbol. But an alternative is an open-vocabulary language model.**

Devise a sensible way to estimate the word trigram probability $p(z|xy)$ by backing off to a letter n-gram model of z if z is an unknown word. Also describe how you would train the letter n-gram model.

We could break up tokens into sub-tokens, which involves splitting words like "unknown" into "un" and "known". For example, if we have $p(z|xy)$, and z is not in our vocab, the probability of getting z could be calculated by first decomposing z into subwords. The subwords could be found from an algorithm such as WordPiece. Then we can train the model on the subtoken sequences.

The formula can start with $z = s_1 s_2 ... s_k$ after running the out-of-vocab string z through WordPiece. From here, we can compute $p(z|xy) = \prod_{i=1}^{k} p(s_i|xy, s_1, ...s_{i-1})$, where we go through all of the subtokens at index i, and each s_i will also take into account the other subtokens.

We could train the model by treating each word in the training corpus as a sequence of subtokens with EOS and BOS tokens to mark the sentence boundaries. We can then estimate probabilities with $p(s_i|s_{i-n+1}^{i-1})$ with appropriate smoothing. This will get us a probability distribution over subtoken sequences rather than the whole word, allowing us to more accurately assess the probability of generating a word, even if it doesn't exist in our vocabulary.