CSE 256 Assignment 3 Report

Vignesh Nanda Kumar: A59010704

The report is about analyzing different n-gram language models. I will be analyzing texts from different domains using the unigram and trigram model with linear interpolation smoothing.

2 Unigram Language Model Analysis

2.1 Analysis on In-Domain Text: For this experiment, I took different proportions of training data for all the 3 datasets and trained the model on the proportion of training data and computed the perplexity on the corresponding train, dev and test datasets. Figure 1 shows the results.

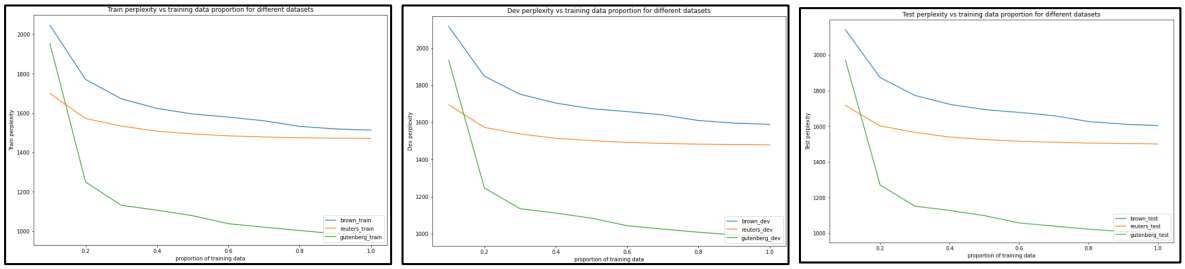

Figure 1: Train, Dev and Test perplexity vs Proportion of data trained on

I have two main observations from the graphs. The first one is that on increasing the proportion of training data, the test perplexity decreases which is because the model learns more with more training data and hence is able to perform better on the test data. The second observation is that Gutenberg dataset has much less test perplexity compared to Brown and Reuters. This is because the total number of training sentences in Gutenberg dataset (68740) is larger than the number of training sentences in Brown (39802) and Reuters (38169). So, with more training data, the training on Gutenberg dataset generates better models.

2.2 Analysis on Out-of-Domain Text: Table 1 shows the perplexity of data "x" obtained using model trained on data "y".

Train	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	1513.80	6780.82	1758.06
Reuters Model	3806.39	1471.21	4882.80
Gutenberg Model	2616.57	12420.13	982.57
Dev	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	1589.38	6675.63	1739.41
Reuters Model	3808.87	1479.09	4833.88
Gutenberg Model	2604.28	12256.3	991.5
Test	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	1604.2	6736.6	1762.01
Reuters Model	3865.16	1500.69	4887.47
Gutenberg Model	2626.05	12392.5	1005.79

Table 1: Perplexity of Unigram model on different datasets

From the results we can see that the Brown model is able to give similar performance (slightly worse) on the Gutenberg dataset for all train, dev and test datasets. But on the Reuters data, the Brown model performs extremely bad. The Reuters model is not able to generalize well to the Brown and the Gutenberg datasets. The Gutenberg model is also performing not that good on the Brown dataset and extremely bad on the Reuters dataset. This trend shows that the Unigram model in general trained on one domain's dataset does not generalize well on another domain's text. This is because the Unigram model does not consider the past context during prediction. The performance is further worsened by the fact that the three datasets belong to different domains and hence can contain different vocabulary. Just the Brown model is able to give slightly acceptable performance on other domains because of its domain being corpus for present day American English (covering majority of the English words).

3 Context-aware LM: Implementation

The language model that I implemented is a Trigram model with linear interpolation based smoothing. For implementing the trigram model, the algorithm I followed is as given:

1) Computing n-gram counts: I first compute the unigram counts to get the vocabulary. Using the unigram counts, I find out the rare words in the corpus (**Hyperparameter: Unknown cut off**). A word is considered rare if the frequency of the word is <= unknown cut off. Rare words are removed from the corpus and their counts are assigned to the "UNK" (unknown) word to handle OOV words. I tried randomly removing some rare words (using a coin flip strategy). All rare words are not removed because then the sentence generation process will generate a lot of UNKs. After the vocabulary is set, I compute the bigram and trigram counts (adding start of sentence * and end of sentence words for simplifying counting).

$$q_{\text{ML}}(w_i) = \frac{\text{Count}(w_i)}{\text{Count}()}$$

Unigram MLE

$$q_{\text{ML}}(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i)}{\text{Count}(w_{i-1})}$$

Bigram MLE

$$q_{\text{ML}}(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

Trigram MLE

$$q(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 \times q_{\text{ML}}(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 \times q_{\text{ML}}(w_i \mid w_{i-1}) + \lambda_3 \times q_{\text{ML}}(w_i)$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i.

Linear Interpolation formulation

Algorithm pseudocode:
1. Compute unigram vocabulary
2. If word frequency <= unk_cutoff:
   - Rare word with probability 0.5
3. Remove rare words from vocabulary
4. Compute bigram and trigram counts
   - Preprocess rare words to UNK
5. Find conditional probability:
   - Preprocess OOV words to UNK
   - Unigram prob = prob if unigram exists else 0
   - Bigram prob = prob if bigram exists else 0
   - Trigram prob = prob if trigram exists else 0
   - Use linear interpolation smoothing to combine probabilities

Algorithm Pseudocode

2) Computing norm: For the linear interpolation smoothing technique, the norm method is not required because we need to first find out the sum in proportion of lambdas and then take logarithm which can be done in the conditional probability computation function. The above equations give the formulation for the linear interpolation method.

3) Computing conditional probability: This function finds out the probability of a word given its past context (previous). Since the linear interpolation method is used, I compute the unigram, bigram and trigram probabilities first. The words and the context words are pre-processed to "UNK" if any word is OOV. For all n-gram probabilities, if the n-gram is present, the probability is non-zero else the probability will be 0. The final probability will be computed using the **lambda values** (hyperparameter) and is given using the below equation. This probability is given to log function for further use.

Using the above algorithm, I implemented the trigram model with linear interpolation smoothing. The hyperparameters in this algorithm are the **lambda values** and the **unknown cut off**.

3.1 Context-aware LM: Analysis of In-Domain Text

To find out the best hyperparameter combination I took up different lambda values ranging from giving unigram model the largest weight to the trigram model being given the largest weight. And the unknown cut off was varied between 1 to 20. I did a grid search over these parameters and evaluating the performance on the dev dataset. The best configuration was found to be lambda values = [0.33, 0.33, 0.34] (equal weights to the three n-gram models) and unknown cut off = 10.0. This makes sense to not have extreme values for lambda because that can give skewed importance to different models. Also unknown value cannot be too high then we will get lot of UNK token during sampling and also it cannot be too low because that leads to bad dev perplexity.

Dataset/Model	Unigram	Trigram Base model	Trigram tuned model
Brown	1604.19	1212.03	**607.57**
Reuters	1500.69	282.42	**209.86**
Gutenberg	1005.78	562.62	**384.93**

Table 2: Test perplexity for different models

Table 2 shows the results of models on the test dataset. It compares the unigram model, trigram base model (with λ1=0.2, λ2=0.3, λ3=0.5, and unknown cutoff = 1.0). The trigram tuned model has hyperparameters: λ1=0.33, λ2=0.33, λ3=0.34, and unknown cutoff = 10.0. I can see that the tuned trigram model with linear

interpolation smoothing gives the best test perplexity. Table 3 and Figure 2 shows the hyperparameter tuning results. For Table 3 experiment, the unknown cutoff was fixed at 10 and for Figure 2, the lambdas were fixed at [0.33, 0.33, 0.34]. We can see that having almost equal lambda (taking equal contributions of n-grams) is giving the best performance for all datasets. And a reasonable unknown cut off to select will be 10 because we don't want a lot of UNK tokens in our sample. But from Figure 2, we can see that on increasing the unknown cut off we get improved dev perplexity because we are adding more rare words in the UNK token set and giving more probability to UNK token.

[λ1, λ2, λ3]	Brown Dev perplexity	Reuters Dev perplexity	Gutenberg Dev perplexity
[0.33, 0.33, 0.34]	**524.54**	**205.90**	**379.19**
[0.1, 0.3, 0.6]	1020.55	269.53	570.84
[0.6, 0.3, 0.1]	592.53	233.82	382.20
[0.3, 0.6, 0.1]	638.82	243.77	432.60
[0.1, 0.6, 0.3]	1008.48	286.82	587.15
[0.9, 0.05, 0.05]	574.28	338.65	457.79
[0.05, 0.9, 0.05]	1479.10	445.35	893.79
[0.05, 0.05, 0.9]	1785.64	374.78	856.43

Table 3: Dev perplexity for different Lambda configurations (unknown cut off fixed = 10.0)

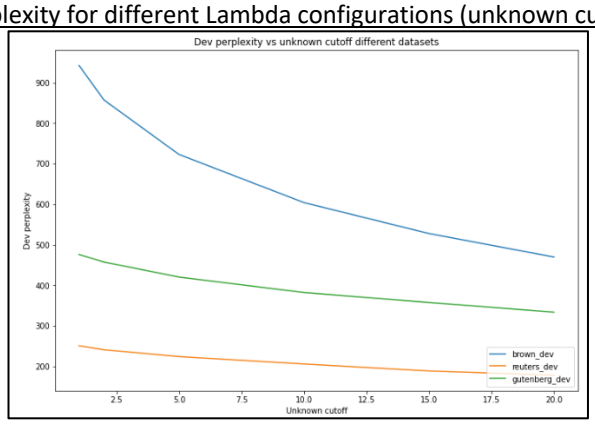

Figure 2: Dev perplexity vs unknown cut off (Lambda fixed : [0.33, 0.33, 0.34])

3.2 Examples of sampled sentences

Trigram Model trained on	Prefix	Sample sentence
Brown	"At the moment"	At the moment was postponed when UNK UNK UNK Oh my God lies on her adequate for double in formally enter into the in UNK of this essay to would in principle already unless green 10 center who missed between went to UNK and * Their heads were in the case **(S1)**
	"The company reported"	The company reported camera five minutes after the holiday
	"Debt of gratitude"	Debt of gratitude body was buried and is 18th green with 69 and three round total of UNK absurdities is * You 26 * * He even makes the UNK war of UNK
Reuters	"At the moment"	At the moment was loss mln vs 26 mln official when provisions for currency Incoming orders 22 pct try the rubber
	"The company reported"	The company reported net loss by 50 grams tonne was 42 cts vs which 14 cts Net 337 000 The banks raised dropped pct to 280 36 billion Avg shrs 74 387 billion stg However minister try to tonne Moore Medical pct owned by **(S2)**
	"Debt of gratitude"	Debt of gratitude vs been Dakota to seven mln Revs 32 mln months does not * MAKES and from 733 one Shr 54 Shr profit saw to Journal last the number of most Continental had operations in UNK MARKET UNK its previously * * The transaction is subject US expansion plans
Gutenberg	"At the moment"	At the moment transgressed he its horsemen bloom say what right as to me when * * Very good so of truth but you De Grey
	"The company reported"	The company reported to have been said of to day thee thy no serious are young And
	"Debt of gratitude"	Debt of gratitude to his companion looking so for is thing of it for earth No more flashing heads of called before the UNK waves pluck it of my UNK down among lands God fool the spiritual put up his coat **(S3)**

Table 3: Examples of sampled sentences

I took 3 prefixes from the 3 datasets using which I formed sentences. It can be seen from the sentences and the log probability of the sentences given the prefixes, the model is generating sentences from the domain it was trained on. The brown data is generating sentences based on general American English. The Reuters data is generating sentences from the financial world and the Gutenberg data is generating sentences as used by prominent English authors. The sentences give an idea of the domain the model was trained on.

Model	Sentence	Log prob
Brown	S1	-336.13
	S2	-520.33
	S3	-439.56
Reuters	S1	-518.30
	S2	-331.27
	S3	-391.37
Gutenberg	S1	-475.14
	S2	-461.24
	S3	-342.86

Table 4: Sentence log probabilities for example sentences

I took 3 sentences from the above sample(S1 generated by Brown model, S2 by Reuters and S3 by Gutenberg model) and found the log probability of the sentences (larger the value, more likely that the model generated the sentence). We can see that for the Brown model S1 has the highest value, for Reuters S2 and for Gutenberg S3. This shows that the models generated sentences from its own corpus with high probability.

3.3 Context-aware LM: Empirical Analysis of Out-of-Domain Text

Train	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	16.31	673.75	676.85
Reuters Model	702.43	15.02	770.43
Gutenberg Model	682.95	808.04	22.09
Dev	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	603.43	673.59	672.36
Reuters Model	698.62	206.56	774.35
Gutenberg Model	677.52	814.39	383.30
Test	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	611.38	669.84	672.80
Reuters Model	707.44	210.87	765.62
Gutenberg Model	679.13	798.52	386.31

Table 5: Perplexity of Trigram model on different datasets

From the above results, we can see that the tuned trigram model with linear interpolation smoothing performs much better than the unigram model and the trigram baseline in the out of domain cases as well. I observe that for the training data, the perplexities have reduced but compared to same domain, the perplexities are high. But when I see the results for the dev and test data, the perplexity values are still closer across domain. The trigram model considering the past context along with linear interpolation smoothing is able to generalize well on out of domain cases.

3.4 Context-aware LM: Qualitative Analysis of Out-of-Domain Text

For the model trained on brown data, it is able to generalize well on Reuters and Gutenberg data because the Brown data covers present day American English. The Reuters data model is still not able to generalize well on Brown and Gutenberg data because Reuters has a very niche vocabulary of financial news and hence that could contribute to its inability to generalize. We can also verify these claims from Table 4. We see that the Reuters model is able to give a good score to its sentence S2 but a very low log probability for S1 and S3 (sentence generated from Brown and Gutenberg models). Also for the Gutenberg model we see that it is not able to generalize well to Brown and Reuters datasets. This is also verified from the sentence log probabilities shown in Table 4. The highest sentence probability is given to the sentence generated using the Gutenberg model (S3).

Another way to look at this is by seeing common words between datasets. We can see that Brown and Gutenberg have the maximum words in common which can attribute to the Brown model generalizing well on the Gutenberg dataset. It can also be observed that Reuters and Gutenberg have the least words in common which is reflected in the perplexity results for those datasets (Reuters not able to generalize well on Gutenberg data and vice versa).

Train	**Brown**	**Reuters**	**Gutenberg**
Brown	-	6790	9000
Reuters	6790	-	5314
Gutenberg	9000	5314	-

Table 6: Number of common words between datasets

4. Adaptation

Train	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	**16.31**	121.06	154.84
Reuters Model	196.07	**15.02**	166.56
Gutenberg Model	217.69	140.38	**22.09**
Dev	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	**603.43**	279.61	355.64
Reuters Model	516.19	**673.59**	394.60
Gutenberg Model	564.99	326.62	**383.30**
Test	**Brown Data**	**Reuters Data**	**Gutenberg Data**
Brown Model	**611.38**	281.33	353.56
Reuters Model	519.09	**210.87**	399.27
Gutenberg Model	568.74	330.17	**386.31**

The fit corpus function is modular to take as input a partial dataset as well. So, once the models are created, I designed a function to give a fraction of training data and I will use that partial data ($3/10^{th}$ of original data size) to adapt the model. Table 7 gives the results obtained on adapting the model on a different data. I tried $1/10^{th}$,$2/10^{th}$ proportions too, but it was with $3/10^{th}$ proportion where I was able to find better performance on dev and test datasets. Train performance will not match because it is not the complete training data but with a small proportion, the model is able to generalize well and able to outperform the dev and test performance when trained on full dataset. To get closer to the performance when training on corpus B's full training set, the proportion of training data used can further be increased till the point the performance is acceptable. This is what I did by starting with $1/10^{th}$ of the training data and increasing from there.

References

- https://pynative.com/python-random-sample/
- Lecture slides
- Linear interpolation smoothing technique