# Assignment #2

CSE 447: Natural Language Processing
Eric Boris

## Written Problems

---

**1**. Consider a simple language in which each token is drawn from the vocabulary $V$ with probability $\frac{1}{|V|}$, independent of all other tokens. Given a corpus of size $M$, what is the expectation of the fraction of all possible bigrams that have zero count? You may assume $V$ is large enough that $\frac{1}{|V|} \approx \frac{1}{|V|-1}$.

- Let $f(V, M)$ be the expectation of the fraction of all possible bigrams that have zero count in terms of the vocabulary V and corpus size M.
- Let $C(i, j)$ be the count of the number of times the token i preceedes the token j.
- Let $\mathbf{1}(\cdot)$ be the indicator function.
- Find $f(V, M)$.

$$
\begin{aligned}
f(V,M) &= \frac{1}{|V^2|}\mathbb{E}\left[\sum_{i,j}^{V}\mathbf{1}\left(C(i,j)=0\right)\right] && \text{There are } |V^2| \text{ permutations of } V \\
&= \frac{1}{|V^2|}\left(|V^2|P\left(C(i,j)=0\right)\right) && \mathbb{E}\left[\textstyle\sum_{i,\,j}^{V}\right] = V^2 \text{ for i.i.d tokens i, j} \\
&= P\left(C(i,j)=0\right) && |V^2| \text{ cancels} \\
&= 1 - P\left(C(i,j)\neq 0\right) && \text{Complement of } P\left(C(i,j)=0\right) \\
&= 1 - \left(M * \frac{1}{|V|} * \frac{1}{|V|-1}\right) && \text{There are M tokens} \\
&= 1 - \left((M-1) * \frac{1}{|V|} * \frac{1}{|V|-1}\right) && \text{There are M-1 bigrams} \\
&= 1 - \left((M-1) * \frac{1}{|V|} * \frac{1}{|V|}\right) && \tfrac{1}{|V|} \approx \tfrac{1}{|V|-1} \\
&= 1 - \left((M-1) * \frac{1}{|V^2|}\right) && \text{Simplify} \\
&= 1 - \frac{M-1}{|V^2|}
\end{aligned}
$$

$$\boxed{f(V,M) = 1 - \frac{M-1}{|V^2|}}$$

---

**2**. Design a feedforward network to compute this function, which is closely related to XOR:

$$
f(x_1, x_2) = \begin{cases} -1 & \text{if } x_1 = 1 \wedge x_2 = 1 \\ 1 & \text{if } x_1 = 1 \wedge x_2 = 0 \\ 1 & \text{if } x_1 = 0 \wedge x_2 = 1 \\ -1 & \text{if } x_1 = 0 \wedge x_2 = 0 \end{cases}
$$

Your network should have a single output node that uses the “sign” activation function,

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \leq 0 \end{cases}$$

Use a single hidden layer, with ReLU activation functions. Describe all weights and offsets.

Define and construct the above described feedforward network as follows.

- Let $\mathbf{x} = [x_1,\, x_2]^{\mathrm{T}}$ be the input vector such that $x_1$ and $x_2 \in \{0,\, 1\}$ are binary scalars.
- Let $\mathbf{\Theta}^{(x\to z)} \in \mathbb{R}^{2\times 2}$ be the hidden layer matrix where $\mathbf{\Theta}^{(x\to z)} = \left[\theta_1^{(x\to z)},\, \theta_2^{(x\to z)}\right]^{\mathrm{T}}$ such that the vectors $\theta_1^{(x\to z)}$ and $\theta_2^{(x\to z)}$ represent the two nodes in the hidden layer.
- Let $\mathbf{b}_1 \in \mathbb{R}^2$ be the hidden layer offset parameter.
- Let $\text{ReLU}\,(a) = \max(0,\, a)$, the standard ReLU function.
- Let $\mathbf{z} = \text{ReLU}\left(\mathbf{\Theta}^{(x\to z)}\mathbf{x} + \mathbf{b}_1\right)$ be the the hidden layer.
- Let $\mathbf{\Theta}^{(z\to y)} \in \mathbb{R}^{1\times 2}$ be the output layer matrix where $\mathbf{\Theta}^{(z\to y)} = \left[\theta_1^{(z\to y)}\right]^{\mathrm{T}}$ such that the vector $\theta_1^{(z\to y)}$ represents the node in the output layer.
- Let $\mathbf{b}_2 \in \mathbb{R}^1$ be the output layer offset parameter.
- Let $\text{sign}(a)$ be the sign function as described above.
- Let $\hat{y} = \text{sign}\left(\mathbf{\Theta}^{(z\to y)}\mathbf{z} + \mathbf{b}_2\right)$ be the predicted output.
- Combining the above yields the feedforward network that predicts $\hat{y}$ as

$$\boxed{\hat{y} = \text{sign}\left(\mathbf{\Theta}^{(z\to y)}\left(\text{ReLU}\left(\mathbf{\Theta}^{(x\to z)}\mathbf{x} + \mathbf{b}_1\right)\right) + \mathbf{b}_2\right)}$$

---

**3**. Consider the same network as in problem 2 (with ReLU activations for the hidden layer), with an arbitrary differentiable loss function $\ell(y^{(i)}, \tilde{y})$, where $\tilde{y}$ is the activation of the output node. Suppose all weights and offsets are initialized to zero. Show that gradient descent will not learn the desired function from this initialization.

- Let $\ell\,(y,\, \tilde{y})$ be an arbitrary loss function.
- Let FFN be the feedforward network from problem 2.
- Suppose all weights and offsets in FFN are initialized to zero.
- With $\eta$ as the learning rate, let the following represent the weight and offset updates via gradient descent.

$$\begin{aligned} \mathbf{\Theta}^{(x\to z)} &\leftarrow \mathbf{\Theta}^{(x\to z)} - \eta\nabla_{\mathbf{\Theta}^{(x\to z)}}\ell \\ \mathbf{\Theta}^{(z\to y)} &\leftarrow \mathbf{\Theta}^{(z\to y)} - \eta\nabla_{\mathbf{\Theta}^{(z\to y)}}\ell \\ \mathbf{b}_1 &\leftarrow \mathbf{b}_1 - \eta\nabla_{\mathbf{b}_1}\ell \\ \mathbf{b}_2 &\leftarrow \mathbf{b}_2 - \eta\nabla_{\mathbf{b}_2}\ell \end{aligned}$$

- Consider each of the above updates.
  - $\mathbf{\Theta}^{(x\to z)} = \mathbf{\Theta}^{(x\to z)}$ because $a = 0$ so $\text{ReLU}(0) = 0$ so $\text{ReLU}'(0) = 0$.
  - $\mathbf{\Theta}^{(z\to y)} = \mathbf{\Theta}^{(z\to y)}$ because $\text{sign}'(a) = 0$ if $a > 0$ and if $a \leq 0$.
- Thus, the combination of zero weights and offsets, ReLU(a), and sign(a), keeps the gradients at zero, and thus the weights and offsets can’t be learned.

# n-Gram Language Modeling

**4.1**. Build and evaluate unigram, bigram, and trigram language models. To handle out-of-vocabulary (OOV) words, convert tokens that occur less than three times in the training data into a special UNK token during training. Provide graphs, tables, charts, or other summary evidence to support any claims you make.

a. Describe your models and experimental procedure.

**n-Gram model**

- Let a **dataset** be a nested list of lists of string tokens.
- Let the **unk threshold** be the minimum number of times that a token $t_i$ appears in a training dataset without being converted into the special UNK token during testing. I.e. if the unk threshold is 3, then any tokens appearing less than 3 times in the training dataset will be converted into the UNK token.
- Let the emission **probability** $P(t_i|t_{i-(n-1)})$ of token $t_i$ for $n \geq 1$ be computed $\frac{C\left(t_i|t_{i-(n-1)}\right)}{|V|}$ where $|V|$ is the size of the vocabulary of the dataset and $C\left(t_i|t_{i-(n-1)}\right)$ is the count of times $t_i$ appears preceeded by $t_{i-(n-1)}$ tokens.
- Let a **unigram** model be a mapping of token $t_i$ to token emission probability $P(t_i)$.
- Let an **ngram** model be a mapping of token $t_i$ to token emission probability $P(t_i|t_{i-(n-1)})$ given $n-1$ preceeding tokens $t_{i-(n-1)}$. We then specify that a **bigram** model is an ngram model mapping $t_i$ to $P(t_i|t_{i-1})$ and that at **trigram** model is an ngram model mapping $t_i$ to $P(t_i|t_{i-2}, t_{i-1})$. Note, that for implementation reasons ngram specifically refers to n>1 ngram models in code, however, for convenience, the term ngram is used in this write up to refer to ngram models where n≥1.
- Combining the above yields the following pipeline for creating an ngram model. Load a training dataset. For each line of tokens in the dataset, unk any applicable tokens. For each token in the line compute it's probability and store both token and probability in the model.

**Perplexity**

- Let the **cross entropy** be $H(T) = -\sum_i^N \frac{1}{N} log_2 P\left(t_i|t_{i-(n-1)}\right)$ where $T$ is the dataset of tokens, $N$ is the number of tokens in the dataset, and $P\left(t_i|t_{i-(n-1)}\right)$ is the probability gotten from a trained ngram model of token $t_i$ being preceeded by tokens $t_{i-(n-1)}$ for $n \geq 1$. Note that in some cases $P\left(t_i|t_{i-(n-1)}\right) = 0$ (ex: a token is observed in the test dataset that had not been observed in the training dataset). In these cases the token is not included in the calculation of the cross entropy to prevent incorrectly skewing the perplexity. Instead, in code, the cross entropy function also returns a percentage of such tokens that were removed from the calcuation.
- Let the **perplexity** be $P(T) = 2^{H(T)}$. If no words in the test dataset $T$ appear in the trained ngram model and $H(T) = 0$ then rather than report a false perplexity, $P(T) = \infty$.
- Combining the above yields the following for computing the perplexity of an ngram model on dataset $T$. For each line in the dataset, replace appropriate tokens with the UNK token. For each token in the line compute it's perplexity $P(T) = 2^{H(T)} = 2^{-\sum_i^N \frac{1}{N} log_2 P\left(t_i|t_{i-(n-1)}\right)}$ and return the total perplexity of the dataset.

b. Report the perplexity scores of the unigram, bigram, and trigram models for your training, validation, and test sets.

Perplexity and Percentage of Tokens Removed by n-Gram models on Training, Validation, and Testing datasets with an unk threshold of 3.

| | Training | | Validation | | Testing | |
|---|---|---|---|---|---|---|
| Model | Perplexity | % Removed | Perplexity | % Removed | Perplexity | % Removed |
| Unigram | 976.54 | 0.00 | 892.25 | 0.00 | 896.50 | 0.00 |
| Bigram | 77.07 | 0.00 | 65.57 | 20.10 | 61.50 | 18.83 |
| Trigram | 7.30 | 0.00 | 11.24 | 56.33 | 10.29 | 54.68 |

c. Briefly discuss the experimental results.

The results show that **increasing n** is correlated with **decreased perplexity** across all datasets but with **increased percentage of words removed** on validation and testing datasets. This is to be expected. Increasing n increases the size of the model which increases the number of permutations of words the model can recognize and thus decrease the model's perplexity to novel phrases. Increasing n increases the size of the model but also the pecularity of a phrase. I.e. suppose a unigram model was trained on the phrase "this and that" and was shown the test phrase "that and this". No tokens will be removed. However if a trigram model is trained on the same phrase and the same test phrase is shown to this model, all the tokens will be removed because the percentage associated with this latter phrase in the trigram model is 0 because this phrase hadn't been seen.

---

**4.2**. Implement linear interpolation smoothing between unigram, bigram, and trigram models:

$$\theta'_{x_j|x_{j-2},x_{j-1}} = \lambda_1\theta_{x_j} + \lambda_2\theta_{x_j|x_{j-1}} + \lambda_3\theta_{x_j|x_{j-2},x_{j-1}}$$

Where $\theta'$ represents the smoothed parameters and the hyperparameters $\lambda_1$, $\lambda_2$, and $\lambda_3$ are weights on the unigram, bigram, and trigram language models, respectively. $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Provide graphs, tables, charts, or other summary evidence to support any claims you make.

a. Describe your models and experimental procedure.

**Linear Interpolation n-Gram Model**

- Given an $n > 1$, build n k-Gram models where $1 \leq k \leq n$ such as those described above.
- For each token $t_i$ in the nth k-Gram model compute the linear interpolation smoothing using the formula described above.

b. Report perplexity scores on training, validation, test sets for various values of $\lambda_1$, $\lambda_2$, and $\lambda_3$. Report no more than 5 different sets of $\lambda$'s. In addition to this, report the training and validation perplexity for the values $\lambda_1 = 0.1$, $\lambda_2 = 0.3$, and $\lambda_3 = 0.6$

Perplexity and Percentage of Tokens Removed by Linear Interpolation Trigram models with different Lambda Parameters on Training, Validation, and Testing datasets with an unk threshold of 3.

| $\lambda_1$, $\lambda_2$, $\lambda_3$ | Training | Validation | Testing |
|---|---|---|---|
| 0.0, 0.1, 0.9 | 7.82 | 43.79 | 43.58 |
| 0.1, 0.3, 0.6 | 10.40 | 289.71 | 288.47 |
| 0.3, 0.4, 0.3 | 16.94 | 230.65 | 230.19 |
| 0.6, 0.3, 0.1 | 36.22 | 257.18 | 257.30 |
| 0.9, 0.1, 0.0 | 282.09 | 437.69 | 440.06 |

The trend is that **increasing $\lambda_3$ decreases perplexity** while **increasing $\lambda_1$ increases perplexity**.

c. If you use half of the training data, would it increase or decrease the perplexity on previously unseen data? Why? Provide empirical experimental evidence if necessary.

Perplexity and Percentage of Tokens removed by n-Gram models after training on half the training dataset on Training, Validation, and Testing datasets with an unk threshold of 3.

| | Training | | Validation | | Testing | |
|---|---|---|---|---|---|---|
| Model | Perplexity | % Removed | Perplexity | % Removed | Perplexity | % Removed |
| Unigram | 816.49 | 0.00 | 723.12 | 0.00 | 725.55 | 0.00 |
| Bigram | 63.08 | 0.00 | 52.09 | 22.07 | 46.24 | 19.71 |
| Trigram | 6.11 | 0.00 | 9.56 | 58.52 | 8.20 | 55.84 |

Perplexity and Percentage Removed percentage differences from training model on a half training dataset from training model on the entire training dataset. Positive values indicate an increase in perplexity from entire training dataset to half training dataset and negative values indicate a decrease in perplexity from entire training dataset to half training dataset.

| | Training | | Validation | | Testing | |
|---|---|---|---|---|---|---|
| Model | Perplexity | % Removed | Perplexity | % Removed | Perplexity | % Removed |
| Unigram | -16.39 | 0.00 | -12.80 | 0.00 | -19.07 | 0.00 |
| Bigram | -18.15 | 0.00 | -20.56 | 9.75 | -24.81 | 4.67 |
| Trigram | -16.30 | 0.00 | -14.95 | 3.89 | -20.31 | 2.12 |

Compared to training the same model son the entire training dataset, **reducing the size of the training dataset** has the effect of **reducing the perplexity** across all models and datasets. However this also has the effect of **increasing the percentage of words removed** since fewer words had been seen during training. Pushed to extremes, we expect that a training dataset of size 0 would therefore remove 100% of words. In conclusion, a balance should be sought between dataset size, percentage of words removed, and perplexity.

d. If you convert all tokens that appeared less than 5 times to <UNK>, would it increase or decrease the perplexity on the previously unseen data compared to an approach that converts only a fraction of words that appeared just once to <UNK>? Why? Provide empirical experimental evidence if necessary.

Perplexity and Percentage of Tokens removed by n-Gram models on Training, Validation, and Testing datasets with an unk threshold of 5.

| | Training | | Validation | | Testing | |
|---|---|---|---|---|---|---|
| Model | Perplexity | % Removed | Perplexity | % Removed | Perplexity | % Removed |
| Unigram | 803.49 | 0.00 | 754.30 | 0.00 | 756.69 | 0.00 |
| Bigram | 75.63 | 0.00 | 64.23 | 18.00 | 60.57 | 16.84 |
| Trigram | 7.95 | 0.00 | 11.61 | 53.74 | 10.67 | 52.08 |

Perplexity percentage difference of training using an unk threshold of 3 compared to using an unk threshold of 5. Positive values indicate an increase in perplexity from unk threshold of 3 to unk threshold of 5 and negative values indicate a decrease in perplexity from an unk threshold of 3 to an unk threshold of 5.

| | Training | | Validation | | Testing | |
|---|---|---|---|---|---|---|
| Model | Perplexity | % Removed | Perplexity | % Removed | Perplexity | % Removed |
| Unigram | -19.44 | 0.00 | -16.76 | 0.00 | -16.94 | 0.00 |
| Bigram | -1.86 | 0.00 | -2.06 | -11.02 | -1.52 | -11.16 |
| Trigram | 8.52 | 0.00 | 3.24 | -4.71 | 3.63 | -4.87 |

Compared to using an unk threshold of 3, **increasing the unk threshold to 5** has the effect of **decreasing** perplexity on **unigram and bigram models** and of **increasing** perplexity on the **trigram model** and of **decreasing** the percentage of words removed across all models. This suggests that there exists a balance to strike between n and the size of unk threshold to minimize perplexity and the percentage of words removed.