Chapter 2

Literature Study

2.1. The Language Model

2.1.1. The Need for the Statistical Language Model

Natural languages spoken by people are unlike programming languages, they are not designed, but rather emerge naturally and tend to change over time. Although they are based on a set of grammatical rules, spontaneous speech often deviates from it. Because of this, trying to program machines to interpret languages can be difficult. Rather than trying to model formal grammars and structures of a language, we find more success in basing them on statistical models. This is far less complex and still allows machines to interpret certain ambiguities, occurring often under natural circumstances.

One such ambiguity can be explained by an example. If a machine's purpose was to translate a user's speech to text, it could encounter the following problem; having to determine which of two words, that share acoustic characteristics, were said by the user. Such as "what do you see?" vs "what do you sea?". For the machine to interpret this correctly, we make use of a statistical language model. These models succeed because words do not appear in random order i.e. their neighbours provide much-needed context to them.

A statistical language model, in short, simply assigns a probability to a sequence of words. By doing so, the machine is able to choose the correct sequence of words based on their probabilities. To be clear, the language model does not assign a probability based on acoustic data, but rather the possible word sequences themselves.

2.1.2. Mathematical Preliminaries

The probability of a sequence of words can be represented as follows:

$$P(\mathbf{w}(0, L-1)) \tag{2.1}$$

with $\mathbf{w}(0, L-1) = w(0), w(1), ..., w(L-1)$ representing a sequence of L words. This probability is used by the machine in the speech-to-text example to successfully distinguish between the ambiguous words and choose the correct sequence. Much like a human uses the context of the given words.

The joint probability in Equation (2.1) can be decomposed, using the definition of conditional probabilities, into a product of conditional probabilities:

$$P(\mathbf{w}(0, L-1) = \prod_{i=0}^{L-1} P(w(i)|\mathbf{w}(0, i-1)) \tag{2.2}$$

Therefore the probability of observing the i^{th} word is dependent on knowing the preceding $i-1$ words. To reduce complexity, we approximate Equation (2.2) to only consider the preceding $n-1$ words:

$$\prod_{i=0}^{L-1} P(w(i)|\mathbf{w}(0, i-1)) \approx \prod_{i=0}^{L-1} P(w(i)|\mathbf{w}(i-n+1, i-1)) \tag{2.3}$$

We have now derived the basis for the n-gram language model using the **Markov** assumption. Markov models are described as probabilistic models that predict some future unit without looking too far into the past [1].

2.1.3. N-Gram Language Model

In this report, we will only be focusing on one type of language model, the n-gram model. The n-gram model has several reasons for its success in becoming one of the preferred models. It considered as being computationally efficient and simple to implement [2].

The next step would be to determine the value of n. This is also referred to as the tuple size. Ideally, we would want a large tuple size, because words could still strongly influence others far down the sequence. However, the number of parameters that need to be estimated grows exponentially as tuple size increases. If we take a vocabulary size of 20,000 words, which is being modest, Table 2.1 shows the number of parameters to be estimated for each tuple size [3].

Table 2.1: Increase of parameters for n-gram size

Tuple Size	Parameters
2 (bigram)	$20,000^2 = 4 \times 10^8$
3 (trigram)	$20,000^3 = 8 \times 10^{12}$
4 (four-gram)	$20,000^4 = 16 \times 10^{16}$
5 (five-gram)	$20,000^5 = 32 \times 10^{20}$

A five-gram model, although having more information available for potentially better estimation of the probabilities, is not computationally practical. Four-gram models are considered to be barely feasible. Models tend mostly to incorporate bigrams and trigrams [3].

2.1.4. Estimating N-gram Probabilities

The simplest way to estimate the probability is with a Maximum Likelihood Estimation, known as MLE. To obtain this estimate, we observe the n-gram counts from a training corpus. These counts are thereafter normalized to give [1]:

$$P_{rf}(w_n|w_1...w_{n-1}) = \frac{N(w_1...w_n)}{\sum_{\forall k} N(w_1...w_{n-1}, w_k)} \tag{2.4}$$

This equation can be further simplified to yield Equation (2.5) as the count of a given prefix is equal to the sum of all n-gram counts containing that same prefix.

$$P_{rf}(w_n|w_1...w_{n-1}) = \frac{N(w_1...w_n)}{N(w_1...w_{n-1})} \tag{2.5}$$

We now have a value that lies between 0 and 1, and will refer to this probability estimate as the relative frequency estimate.

A new problem arises because our training set and validation set have not necessarily seen the same n-grams. Even if we increase the size of the training set, the majority of words are still considered to be uncommon and n-grams containing these words are rare [3]. The validation set will most likely contain several unseen n-grams whose relative frequencies according to Equation (2.5) is considered to be 0.

The probability of occurrence for a sequence of words should never be 0, however since all legitimate word sequences will never be seen in a practical training set, some form of regularization is needed to prevent overfitting of the training data. This is where smoothing

is applied. Smoothing decreases the probability of seen n-grams and assigns this newly acquired probability mass to unseen n-grams [3]. The next section will discuss different kinds of smoothing and how they are implemented.

2.2. Smoothing

2.2.1. Good-Turing

This smoothing technique was published by I. J. Good in 1953, who credits Alan Turing with the original idea [4]. It is based on the assumption that the frequency for each, in our case, n-gram follows a binomial distribution [1]. Specifically, n-grams occurring the same number of times are considered to have the same probability estimate.

We begin with the Good-Turing probability estimate:

$$P_{GT} = \frac{r^*}{N_\sum} \tag{2.6}$$

Where $N_\sum$ is the total number of n-grams and r^* is a re-estimated frequency formulated as follows [3]:

$$r^* = (r+1)\frac{E(C_{r+1})}{E(C_r)} \tag{2.7}$$

In Equation (2.7) C_r refers to the count of n-grams that occur exactly r times in the training data, and $E(\cdot)$ refers to the expectation. By approximating the expectations as counts and combining Equations (2.6) and (2.7) we obtain:

$$q_r = \frac{(r+1)\cdot C_{r+1}}{N_\sum \cdot C_r} \tag{2.8}$$

In Equation (2.8), q_r denotes the Good-Turing estimate for the probability of occurrence of an n-gram occurring r times in the training data. For unseen n-grams, $r = 0$, giving us:

$$C_0 \cdot q_0 = \frac{C_1}{N_\sum} \tag{2.9}$$

and where $C_0 \cdot q_0$ can be considered as the probability of occurrence of any unseen n-gram.

The Good-Turing technique can be applied to a language model, in order to provide nonzero probability estimates for unseen n-grams. However, this technique has two concerns.

One is that our assumption in Equation (2.8) is only viable for values of $r < k$ (where k is a threshold, typically a value ranging from 5 to 10 [1, 3]). The MLE estimate of high

frequency words is accurate enough that they do need smoothing.

The other concern being that it is quite possible for C_r to equal 0 for some value of r. This would result in a probability estimate of zero, leading us back to the start of our problem. To remedy this, one can smooth the values for C_r, replacing any zeroes before calculating the probability.

2.2.2. Backing-Off

Instead of redistributing the probability mass equally amongst unseen n-grams, S. M. Katz suggests an alternative. His solution, introduced in 1987 [1], approximates a nonzero probability estimate to unseen n-grams, by *backing off* to the $(n-1)$-gram. This backoff will occur until a n-gram with a nonzero count is observed. By doing so, the model is able to obtain more reliable information from an unseen n-gram and use this to better estimate its probability of occurrence.

In Katz's backoff model a discounting factor reserves probability mass. This discounting factor could be implemented in the form of absolute discounting, linear discounting or other suitable estimators. Thereafter backoff dictates the redistribution of this probability mass amongst unseen n-grams [1].

We will be combining Katz's backoff model with Good-Turing discounting. Our combined model is described as follows:

$$P_{bo}(w_n|w_1...w_{n-1}) = \begin{cases} P^*(w_n|w_1...w_{n-1}) & \text{if} \quad N(w_1...w_{n-1}) > 0 \\ \alpha(w_1...w_{n-1}) \cdot P_{bo}(w_n|w_2...w_{n-1}) & \text{if} \quad N(w_1...w_{n-1}) = 0 \end{cases} \tag{2.10}$$

where P^* is the Good-Turing discounted probability estimate and $\alpha(w_1...w_{n-1})$ is the backoff weight introduced by Katz. The second case in Equation (2.10) shows how we can recursively back off in the presence of an unseen n-gram, obtaining a nonzero probability estimate that has been normalised by the backoff weight.

It is important to use discounted probability estimates to prevent adding probability mass into the system during backoff. To further ensure true probabilities are produced, the backoff weight is chosen to satisfy the following requirement [1];

$$\sum_{\forall k} P_{bo}(w_k|w_1...w_{n-1}) = 1 \tag{2.11}$$

Thereby normalising the probability estimate $P_{bo}(\cdot)$. In Equation (2.11) k represents the number of n-grams in our training set containing the same prefix.

To derive a the value for $\alpha(w_1...w_{n-1})$, we start by defining $\beta(w_1...w_{n-1})$ as the harvested probability mass obtained during discounting. β is calculated by subtracting from 1 the total discounted probability mass for all n-grams, seen in our training set, sharing the same prefix:

$$\beta(w_1...w_{n-1}) = 1 - \sum_{\forall k:r>0} P^*(w_k|w_1...w_{n-1}) \tag{2.12}$$

with r indicating the exact number of occurrence for the n-gram in the training set.

We normalise Equation (2.12) to ensure each $(n-1)$-gram only receives a fraction of the total mass. The normalising factor is calculated by summing the probability estimate of all $(n-1)$-grams sharing the prefix as the unseen n-gram:

$$\gamma(w_1...w_{n-1}) = \sum_{\forall k:r=0} P^*(w_k|w_2...w_{n-1}) \tag{2.13}$$

Which is more conveniently expressed as:

$$\gamma(w_1...w_{n-1}) = 1 - \sum_{\forall k:r>0} P^*(w_k|w_2...w_{n-1}) \tag{2.14}$$

where r represents the exact number of occurrence for the original n-gram (not to be confused with the $(n-1)$-gram) in the training set. Combining Equations (2.12) and (2.13) we find:

$$\alpha(w_1...w_{n-1}) = \frac{1 - \sum_{\forall k:r>0} P^*(w_k|w_1...w_{n-1})}{1 - \sum_{\forall k:r>0} P^*(w_k|w_2...w_{n-1})} \tag{2.15}$$

In our case, we substitute values for n in Equations (2.10) and (2.15), yielding our bigram and trigram models:

$$P_{bo}(w_2|w_1) = \begin{cases} P^*(w_2|w_1) & \text{if} \quad N(w_1,w_2) > 0 \\ \alpha(w_1) \cdot P^*(w_2) & \text{if} \quad N(w_1,w_2) = 0 \end{cases} \tag{2.16}$$

with

$$\alpha(w_1) = \frac{1 - \sum_{\forall k:r>0} P^*(w_k|w_1)}{1 - \sum_{\forall k:r>0} P^*(w_k)} \tag{2.17}$$

and

$$P_{bo}(w_3|w_1,w_2) = \begin{cases} P^*(w_3|w_1,w_2) & \text{if } N(w_1,w_2,w_3) > 0 \\ \alpha(w_1,w_2) \cdot P_{bo}(w_3|w_2) & \text{else if } N(w_2,w_3) > 0 \\ \alpha(w_1,w_2) \cdot \alpha(w_2) \cdot P^*(w_3) & \text{otherwise.} \end{cases} \tag{2.18}$$

with

$$\alpha(w_1,w_2) = \frac{1 - \sum_{\forall k:r>0} P^*(w_k|w_1,w_2)}{1 - \sum_{\forall k:r>0} P^*(w_k|w_2)} \tag{2.19}$$

Where $P^*(w)$ represents Good-Turing discounted probability estimate of that unigram. The last case in Equation(2.18) indicating a further backoff if the $(n-1)$-gram is also unseen.

Backoff models do exhibit a flaw under certain circumstances. Take the trigrams $w_i w_j w_k$ for example. The bigram $w_i w_j$ and unigram w_k could both have high counts in our training set, however there could be a significant reason, possibly grammatical, for these words not appearing as a trigram. The backoff model, being very simple, is not capable of distinguishing this and will assign it a probability estimate, most likely too high. This taken into account, backoff models have proven to work well in practise [1, 3].