SUBSEQUENCE SIMILARITY LANGUAGE MODELS
Juan M. Huerta
IBM T. J. Watson Research Center, Yorktown Heights, NY U.S.A
huerta@us.ibm.com

Abstract
In this work we present the Subsequence Similarity Language Model (S2-LM) which is a new approach to language modeling based on string similarity. As a language model, S2-LM generates scores based on the closest matching string given a very large corpus. In this paper we describe the properties and advantages of our approach and describe efficient methods to carry out its computation. We describe an n-best rescoring experiment intended to show that S2-LM can be adjusted to behave as an n-gram SLM model.

Index Terms: language models, longest common subsequence

1. Introduction
In this work we introduce the Subsequence Similarity Language Model (S2-LM) which, similarly to its traditional n-gam based counterpart, is intended to associate quantitative scores to sentences (or strings, in general) given a string corpus. However, unlike n-gram approaches, S2-LM is based on the string similarity between a target sentence and the set of closest matching strings in a corpus. Because the fundamental differences between n-gram and S2-LM, we believe that our approach presents characteristics that might be in some cases complementary to those of n-grams in specific speech and NLP tasks. This paper is the continuation of our previous work described in [4] and [5].

1.1. Sentence Feasibility & Complete Corpora
The basic motivation behind S2-LM is that as the size of a language corpus grows towards infinity, the corpus is expected to eventually contain all the practically possible (i.e., feasible) sentences in that language. In other words, a sentence observed in such ideally complete corpus is the evidence of its feasibility. Feasibility is thus a concept different from the frequentist or probabilistic foundation of n-gram LMs. A sentence is feasible regardless of how many times it has been observed in this complete corpus.

On the other hand, finite corpora can be seen as a sampling of an ideally large corpus of feasible sentences, i.e., a sample of the language. Under certain distributional assumptions, the probability of a sentence being feasible in an unobserved complete corpus can be assumed to be inversely proportional to the distance (under some normalized string distance function) to the closest match in the incomplete corpus.

1.2. Contributions of this paper
The specific contributions of this paper are:
• We introduce and define S2-LM and compare it versus traditional n-gram SLM (sec. 2)
• We describe a method for the large-scale approximate subsequence similarity computation including some practical heuristics (sec. 3)
• We characterize the computational complexity of the large-scale S2-LM algorithm (sec. 4)
• We provide an overview of related work (sec. 5)
• We show a set of experiments that demonstrate that S2-LM can be adjusted to behave as an n-gram based SLM model of arbitrary order (sec. 6).

2. S2-LM
2.1. Traditional N-gram Language Model
N-gram based statistical language models are used basically to compute the probability of a target sentence assuming:

\[ p(w_1...w_j) = P(w_j | w_{j-1}) \times \cdots \times P(w_1 | w_0) \]

Assuming also that only the most recent context influences the current word and focusing on trigrams leads to:

\[ P(w_1...w_j) = \prod_{i=1}^{j} P(w_i | w_{i-1}, w_{i-2}) \]

An n-gram probability is defined as the number of counts an n-gram is observed in a non-compact corpus (see 2.2) divided over the number of n-grams in the corpus.

2.2. Definition: Compact Corpora
A corpus typically contains phrases and sentences occurring multiple times. Keeping all these naturally occurring multiple instances is important in order to faithfully reflect frequency information needed in the n-gram approach. Since, in principle, S2-LMs are meant to be agnostic to frequency information sentence/phrase replicates can be removed from the corpus thus resulting in a more parsimonious sample of the language. In this paper we call this type of corpus a compact corpus.

2.3. Basic S2-LM Algorithm
Computationally, the S2-LM algorithm consists of two steps
• In the first step, given a compact corpus \( \Psi = \{B_1, B_2 ... B_k\} \) and a target sentence \( A \), we identify the k-closest matching sentences (or sentence segments) in the corpus based on a subsequence similarity distance. While this operation can be carried out exhaustively (sec. 2.6), it can also be carried out more efficiently based on algorithms like the one described in sec. 3 which uses a stack decoder and a positional inverted index.
• In the second step we compute a score based on a function of the distances or similarity scores from the target \( A \) to the top k-hypotheses. Depending on the application, the desired score can be based on normalized string similarity, produced through a regression, etc.
2.4. S2LM vs. n-gram SLM

N-gram SLMs have characteristics that are fundamentally different from S2-LMs:

1. Inherent sentence length penalization: Because the n-gram probabilities are always \( \leq 1 \), the longer a sentence is, the more n-grams the sentence will have and the lower its probability will be (i.e., lower than its suffix). Longer sentences are thus naturally penalized. This is not the case with S2-LM: if a sentence has been observed verbatim in the corpus, given a perfect search algorithm it should be given a higher score than another sentence that has been observed partially, regardless of length.

2. Limited long term and non-contiguous modeling: n-grams have limited ability to model context beyond their order (e.g., 3-grams are limited to two words of context). Because of S2-LMs ability to extend the search computation in an arbitrary order and span (sec. 3.4) it has better theoretical ability to model longer and non-contiguous contexts.

3. Unpopular viewpoint penalization: n-gram SLMs will naturally increase the score of a phrase occurring many times over the score of another less popular phrase. Yet both sentences might be equally correct and feasible. In some applications it might be reasonable to avoid the natural penalization that the less popular phrase receives.

2.5. Underlying similarity functions: SED & LCS

We now provide an overview of the 2 fundamental string similarity functions useful to S2-LM with the goal of explaining decomposition necessary for large scale S2-LM computation. The String Edit Distance, or SED [8], between sentences \( A \) and \( B \) consists of the sum of the cost of the edit operations under the optimal alignment \( \gamma \) (where the cost of the \( k^{th} \) insertion, deletion or substitution are \( \phi_k \), \( \psi_k \) and \( \xi_k \), respectively):

\[
\text{SED}(A, B) = \min_{\gamma} \left( \sum_k \phi_k + \sum_{i,j} \psi_k + \sum_{i,j} \xi_k \right)
\]

For large-scale computation (when \( |\Psi| \) is very large), we will see in later sections that it is more natural to approximate the Longest Common Subsequence between string pairs. The LCS relates to the string edit distance as follows:

\[
\text{SED}(A, B) = n + m - 2|\text{LCS}(A, B)|
\]

where \( n \) and \( m \) denote the length of \( A \) and \( B \) in words. When \( n = m \) for every sentence \( B \) in the corpus, then LCS and SED are bounded and related by a negative scaling factor. However, this situation \( (n = m) \) is generally not expected to be the case when working with large natural language corpora.

2.6. Naive String Similarity Computation

Using dynamic programming (DP) one can simultaneously obtain the optimal alignment and the associated minimum total distance between \( A \) and \( B \). The computational cost of naive SED using DP is \( O(nm) \), where \( n \) is the length of string \( A \) and \( m \) is the length of string \( B \). When the comparison is performed between string \( A \) and every sentence in a corpus \( \Psi = \{B_1, B_2, \ldots, B_k\} \) the cost of exhaustively computing SED using naïve DP is \( O(|\Psi|nm) \). In this case \( m \) is the average sentence length in \( \Psi \), and \( |\Psi| \) is the set cardinality. \( |\Psi| \) is typically very large (many millions of sentences) while \( n \) and \( m \) are relatively small ([8]).

3. Large Scale Computation

The goal of S2-LM is to find, given a target sentence, the closest sequence in a large corpus. We now describe a method to carry out this search using an approximation of the SED.

3.1. The use of an inverted index \( Y \) of \( \Psi \)

The large scale computation of \( \text{SED}(A, B_j) \) is carried out using information represented in an inverted index \( Y \) in which each word instance in the corpus \( \Psi \) is represented. Thus, for each word \( a_i \) in the target sentence \( A \) we will consider each instance the evidence set \( \text{Xai} = \{a_1, a_2, \ldots, a_s\} \subset Y \) and keep track of this evidence using a stack (sec 3.4)

3.2. SED Decomposition

In order to carry out the large scale computation of the SED using the evidence sets contained in the inverted index \( Y \) it is useful to transform the SED score from a distance into a similarity score. For this we define the non-negative complemented SED (denoted as SED* ) between \( A \) and the \( j^{th} \) sentence \( B_j \) in \( \Psi \):

\[
\text{SED}^*(A, B_j) = n - \text{SED}(A, B_j)
\]

When \( A = B_j \) then the SED* in terms of two components: one provided by the counts directly identified using the index (equal to \( n \) minus deletions). We call this the observable evidence (the terms accounted for in the index, as described in 3.1). The other term arises from the total number of spurious terms (\( n \) minus insertions). This is the unobservable terms (the terms not accounted for in the index). Then:

\[
\text{SED}^*(A, B_j) = (n - \sum_{i \in \text{Xai}} \psi_i) + (n - \sum_{i \in \text{Xai}} \xi_i - \sum_{i \in \text{Xai}} \phi_i)
\]

When estimating SED* for a collection of strings and a query we keep track and update the SED* tallies for each sentence in \( \Psi \) reflecting the sum of the observable and unobservable evidence as each of the terms \( \alpha_i \) in the query \( \{a_1, a_2, \ldots, a_s\} \) is incrementally considered. We denote the observable evidence (\( n \) minus deletions) up to term \( a_i \) as \( g(i) \). While \( h(i) \) denotes the estimate up to term \( a_i \) of unobservable terms (amortized deletions). Thus, the SED* up to the \( i^{th} \) term is:

\[
\text{SED}^*_{i}(A, B_j) = g(i) + h(i)
\]

This breakdown will be useful when we implement the A* search algorithm [11]. Now, we want \( g(n) \) to approximate the actual insertion related score as:

\[
g(n) = n - \sum_{k \in J} \psi_k
\]

As the evidence is incrementally introduced we update \( g(i) \):

\[
g(i) = g(i-1) + \Delta g_i
\]

The incremental contribution of term \( a_i \) to \( g(i) \) is computed based on the linear distances between the position of that term and the previously considered term \( a_{i-1} \) in the query and hypothesis:

\[
\Delta g_i = 1 - \text{dist}_{A}(a_i, a_{i-1}) - \text{dist}_{B_j}(a_i, a_{i-1})
\]

The incremental contribution of a term should only be accounted if its location in the sentence is consistent with the hypothesis in question and the previously observed evidence. In the case of \( h(i) \) the increment depends on the amortized
expected deletions is: \( \Delta_{i} = 1 - \frac{m - |B_{i} \cap A|}{|B_{i} \cap A|} \)

Where \( |B_{i} \cap A| \) denotes the cardinality of the set of terms common in the hypothesis and in the query, and \( m \) is \( |B_{i}| \).

3.3. LCS based computation

As described in [10], for certain cases of cost values, there is equivalence between LCS and SED. When this is the case we can equivalently express the following decomposition as follows:

\[ LCS^*(A, B_{j}) = g(i) + h(i) \]

The incremental contribution of \( g \) remains the same:

\[ \Delta_{i} = 1 - \frac{m - |B_{i} \cap A|}{|B_{i} \cap A|} = 1 - m = \text{const}. \]

Because of this term being constant across competing hypotheses, its effect can be ignored during search.

3.4. StackLCS*: A* Stack Decoder Algorithm

In the previous section we described an approach to compute and handle the tallies or scores of competing hypotheses when searching for the best string match in a corpus given a query. We now propose the practical use of \( A^* \) search [9, 6] to conduct best-first tree search using the incremental large scale LCS* approximation. In this algorithm, hypotheses are introduced and/or extended as the evidence from the index is considered. The algorithm, StackLCS*, is outlined in Figure 1 below. But first, we present some basic heuristic extensions.

3.5. Heuristic Extensions

- **Evidence Sampling**: If the evidence set \( X_{ai} \) is very large, it can result in the inability by stack to handle it efficiently (overflow). We propose using a random subset of \( X_{ai} \). This is equivalent to subsampling the corpus and finding the top scoring hypotheses in this subspace. A few sampling iterations can be carried out. Subsampling is better used together with non-compacted corpora.

- **Span Constraining**: Only terms in the evidence list \( X_{ai} \) that are within a certain distance to a hypothesis \( j \) are to be considered. This limits the computational complexity and provides bounds on the modeling span.

- **Randomized insertion order**: Change the order in which the query terms are taken into consideration (starting from most infrequent words). While the order in which the words are considered changes, the distances or relationships between these words should not change.

- **Non-linear Distance function**: Use a non-linear distance function like:

\[ \Delta_{i} = 1 - \frac{\alpha \cdot \text{dist}_{j}(a_{i}, a_{j}) + \beta \cdot \text{dist}_{i}(a_{i}, a_{i-1})}{\text{const}.} \]

Where \( \alpha \) and \( \beta \) are empirically determined and \( r_{i} \) denotes the rank of word \( a_{i} \) when the sentence is sorted by rarity. This helps attenuate the contribution of more common (i.e., stop) words and adjust the increment based on relative distances.

3.6. N-gram & S2LM scores & perplexity

In section 1.1 we discussed the case of a finite corpus, and how the distance to the closest segment could be used to find an approximation to the n-gram SLM score. Pushing this idea further, one can intentionally subsample a corpus with the goal of finding a small set of segments (i.e., a basis) that can be used to carry out SLM scores approximation. As a trivial choice, the set of n-grams in an n-gram LM can be chosen as a basis. Yet how to non-trivially achieve this is the focus of future work.

Given an LM and an unobserved test set, perplexity tells us how well the LM approximates entropy of the test set. Perplexity, thus, only makes sense for S2LM when working with incomplete corpora situations. In this case, perplexity can be associated with how well the segments in the S2LM approximate the test set sentence density in the sentence-space. How to calculate this is also a focus of future work.

Input: query string \( A = \{a_{1}, a_{2}, ..., a_{k}\} \) and inverted index \( Y \)

Output: top hypothesis score \( s \)

do \( K \) times:

create stack \( S \)

for each word \( a_{i} \) in \( A \) do in random order \( \{a_{i}, a_{i+1}, ..., a_{k}\} \)

obtain the set of sentences \( X_{ai} \) of length \( a_{i} \) from \( Y \)

subsample \( X' \subseteq X \)

for each sentence \( B_{j} \) in set \( X' \) do

if \( B_{j} \) exists in \( S \) and Span Constrains are met

compute \( \Delta_{i} \) and update hypoth. score and positions

else

append \( B_{j} \) sentence id to \( S \) with score -1 and initial positions

eliminate hypotheses with score smaller than \( P\% \) of top score

sort stack \( S \) by score and obtain top score for iteration: \( s_{i} \)

return Average Language score \( \pi = \frac{1}{K} \sum s_{i} \)

Figure 1 Algorithm: StackLCS* with Heuristics

4. Computational Complexity

We said in sec. 2.6 that the naïve complexity of SED is \( O(|W| \cdot m) \). It can be shown that the complexity of the StackLCS* shown in figure 1 is \( O(|W| \cdot m \cdot \zeta) \) where \( \zeta \) denotes the average number of operations needed to insert a hypothesis in the stack and \( \Delta \) is the size of the dictionary (which typically is many orders of magnitude larger than \( \gamma \)). Thus StackLCS* is much faster than naïve LCS or SED computation (this was empirically confirmed in [4]).

5. Related Work

Previous authors have used information retrieval and string similarity approaches in NLP and ASR. [3] used LCS to evaluate the quality of MT output given a set of references. [1] used string alignments to learn paraphrasing rules and patterns. LMs have been used both to support IR as a method to extend queries or to obtain relevant document with which new n-gram language models are trained [7] in this paper we focus on using IR to carry out language modeling. There is also work regarding whole sentence LMs [13] that, while not based on SED can be thought of as a way to overcome n-gram limitations.
6. Evaluation

In previous work [4,5] we evaluated the ability of a stack approach to recall an optimal match given a query, as well as the speed compared to naïve SED. We now want to investigate the question of n-gram equivalence, namely, to How does S2LM behave as compared to n-gram SLM in a LM rescoring task? To address this question we built n-gram LMs of different orders using a large but finite text corpus. Using the same corpus we built an inverted index and configured an S2-LM model. Using a test set we identified segments of different syntactical categories and constructed n-best lists using an approach described below. Finally we rescored the n-best lists using the different LMs and counted the number of times that the sorted n-best matched the ground truth in each LM and category configuration.

The LM data: Spinn3r Blog Corpus: Our language models are built using the blog corpus provided by Spinn3r [2]. This corpus consists of 44 million blog posts of variable length that originated between August and September 2008 from which we selected, cleaned, and normalized 2 million blog posts containing 256 million words. Using the Spinn3r corpus we built Kneser-Ney smoothed 2-gram, 3-gram, 4-gram and a 5-gram statistical language models using the SRILM toolkit [12]. We also created an inverted index with word position information from this data based on a 20,000 word dictionary. The data of September 30 was held out as from the training corpus and used as a test set. From it 200 sentences were further sub-selected for development and parameter tuning.

Evaluation Task: Our task consists of using the LMs we built to score n-best lists. Each n-best list has an error free sentence and a sentence with errors and is sorted error-free first. We compared the rank orders of the sentences in each n-best list as ranked by the LMs against the ground-truth-ordering and counted how many times each LM type got the correct order in each sentence category (we had 4 sentence categories, see below). The idea is to model the underlying task of an LM in selecting error-free hypotheses from a set of ASR or MT hypotheses which can be thought of error-free sentence and a noisy sentence and a noisy occurring in the corpus. We call this the natural sentence and a noisy version of it in each n-best list; both of these with the same length. Word mutations were obtained by substituting 3-grams a-b-c with a-y-c where \( p(a-b-c) > p(a-y-c) \) in the corpus.

Evaluation Data: We created 22,000 N-best lists. As mentioned, each n-best list has a natural sentence and a mutation. Depending on the type of natural sentence in N-best list, the list falls into one of the following 4 categories: Category 1. Full sentence any length. Category 2. Random segment of length 6. Category 3. Noun Phrase extracted from parsed sentences (using the factored parser of Stanford Parser [14]). Category 4. Noun Phrases of length 6.

S2-LM Configuration:: We limited the context span to \( \leq 4 \). We sub-sampled instance sets proportionally to the stack space available. The stack depth used is 1600 and after each term insertion in the stack we pruned the stack to 800. We also used the non-linear distance function described in 3.5 and used an overlapping sliding analysis window of length 6 with the final score being the sum of the log scores of the segments.

Results: Figure 2 shows curves where the ranking accuracy is reported for various n-gram orders and for our S2-LM. We can see that as we increase the n-gram order, the accuracy of an SLM increases consistently. This is a reasonable indication that the larger orders of the SLM (especially \( \geq 2 \)) are useful in identifying the word mutations introduced. We can see that the behavior of S2-LM is consistently better than the bigram and in categories 2, 3 and 4 better than the 3-gram even. In general it ranges between bigrams and five-grams with a typical behavior of a 4-gram LM which is consistent with the span used.

7. Conclusion

We presented a new approach to language modeling based on string similarity and described an approach to carrying out the large scale computation of the S2-LM. In our rescoring experiments we saw that our approach can produce results consistent with n-gram models consistent with the S2-LM analysis span. Applications of our approach include, among others, situations in which whole sentences need to be evaluated according to their feasibility without penalizing for length, situations dealing with extremely large corpora, situations where long non-contiguous context is needed, etc.

8. References