Predicting Alignment Distances via Continuous Sequence Matching

Jian Chen[1*] Le Yang[1*] Lu Li[1]

Yijun Sun[1,2,3†]

[1]Department of Computer Science and Engineering
[2]Department of Microbiology and Immunology
[3]Department of Biostatistics
University at Buffalo, The State University of New York
Buffalo, NY 14203

Abstract

Sequence comparison is the basis of various applications in bioinformatics. Recently, the increase in the number and length of sequences has allowed us to extract more and more accurate information from the data. However, the premise of obtaining such information is that we can compare a large number of long sequences accurately and quickly. Neither the traditional dynamic programming-based algorithms nor the alignment-free algorithms proposed in recent years can satisfy both the requirements of accuracy and speed. Recently, in order to meet the requirements, researchers have proposed a data-dependent approach to learn sequence embeddings, but its capability is limited by the structure of its embedding function. In this paper, we propose a new embedding function specifically designed for biological sequences to map sequences into embedding vectors. Combined with the neural network structure, we can adjust this embedding function so that it can be used to quickly and reliably predict the alignment distance between sequences. We illustrated the effectiveness and efficiency of the proposed method on various types of amplicon sequences. More importantly, our experiment on full length 16S rRNA sequences shows that our approach would lead to a general model that can quickly and reliably predict the pairwise alignment distance of any pair of full-length 16S rRNA sequences with high accuracy. We believe such a model can greatly facilitate large scale sequence analysis.

*Equal contribution
†Corresponding Author. Email: yijunsun@buffalo.edu

1 Introduction

The comparison of biological sequences is probably the most important research area in the field of bioinformatics. Sequence comparison algorithms have been widely used as the basis for various applications, such as phylogenetic analysis [1, 2], *de novo* genome assembly [3, 4] and taxonomic classification [5]. In recent years, the sequences used for comparison have changed dramatically. On the one hand, the amount of sequence data has increased significantly. As the second-generation high throughput sequencing technology matures, we have accumulated a large number of sequences and the total amount of sequences is still growing at a rate of (1021) bps each year [6]. On the other hand, the length of sequences is also greatly increased. Compared with the second-generation sequencing technology, the third-generation sequencing technologies such as single-molecule real time (SMRT) sequencing from Pacific Biosciences (PacBio) [7] and nanopore-based sequencing from Oxford Nanopore Technologies [8] has greatly increased the length of raw sequences, from hundreds of bps to tens of thousands, even millions of bps [9]. Although a large number of sequences and long sequence lengths provide us with the opportunity to obtain more accurate results in many applications [4, 5], they also place higher requirements on the accuracy and speed of sequence comparison algorithms.

In this paper, we focus on pairwise sequence comparison, the purpose of which is to measure the similarity or distance of a pair of sequences based on the similarity regions shared by the two sequences. The central problem is that insertions/deletions in a sequence can make it difficult to identify similar regions. The classic Needleman-Wunsch (NW) algorithm [10] solves this problem by searching for the optimal alignment between two sequences using dynamic programming. The NW algorithm can accurately capture evolutionary relationships and is often considered a reference method [11]. However, with the increase in the number of sequences and the increase in sequence length, its high computational complexity makes it impossible to apply in practice. Therefore, in recent decades, researchers have developed a large number of sequence comparison algorithms to adapt to the increase in the number and length of sequences by avoiding the alignment process [11]. For example, the classic k-mer method [12] represents a sequence as a vector using consecutive words of length k and then define the similarity between two vectors as the similarity of the original sequences. Due to its fast speed, this type of method can be used for longer sequences and has become the basic starting point for many alignment-free methods. Also, methods such as kmacs [13] use the statistics of the common subsequences to define the similarity, which can also achieve a reasonable speed for largescale analysis. However, these methods cannot replace the alignment-based method in the ability to describe evolutionary relationships.

Recently, a data-dependent alignment-free method called SENSE [14] was proposed to estimate the alignment distance at a speed close to that of the traditional alignment-free methods. The main idea is to train the neural network model with the supervision of true alignment distance so that the trained model can predict the alignment distance of unseen sequences. Experiments show that SENSE can efficiently and accurately estimate alignment distances for sequences from two sub-regions of the 16S rRNA gene. However, SENSE is only a preliminary attempt and has many weaknesses. First, it cannot be applied to sequences of varying lengths. Second, due to the structure of its embedding function, SENSE cannot handle insertion/deletion, which is very common in biological sequences. Finally, we will see in the experiments that even for equal-length sequences, SENSE does not outperform traditional alignment-free methods such as kmacs on some datasets.

To solve the above issues, we propose an embedding function specifically designed for biological sequences, referred to as the continuous sequence matching (CSM) function. The main idea is to construct the embedding vector of an input sequence by matching it with a set of short kernel sequences. Besides, we took the insertions/deletions of biological sequences into consideration to make the matching results more robust. To assess the effectiveness and efficiency of the proposed method, we performed experiments on various types of amplicon sequences and compared them with five competing methods. The results show that our method outperformed SENSE and other traditional alignment-free methods on all datasets. Moreover, by comparing the CSM function with its gap-free version, we showed that introducing gaps into the model can make the results robust to sequence changes. More importantly, the results on full-length 16S rRNA sequences suggested that our method will lead to a universal model that can quickly and reliably predict the pairwise distance of any full-length 16S rRNA sequence pairs. We believe that this model can greatly facilitate large-scale sequence analysis.

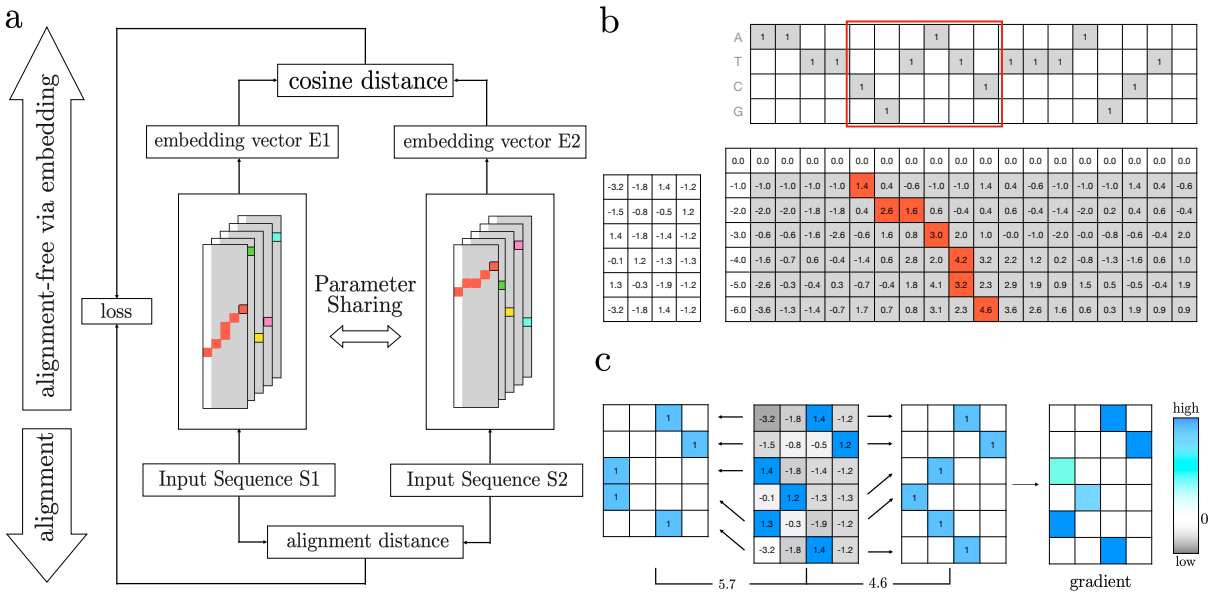


Figure 1: Overview of proposed CSM model and its training process. (a) Siamese network structure and the CSM function with a set of continuous 4-D sequences as learnable parameters. (b) The continuous sequence matching process for one kernel sequence. The red square outlined the best matching subsequence and the red entries in the table indicate the optimal alignment path. (c) The parameter sequence get similarity score and gradient from both best matching subsequences from the input pair using the corresponding optimal alignment. Then combine the gradients using the gradient flow from the cosine-distance function and the loss function

2 Methods

We developed a neural network model, referred as Continuous Sequence Matching (CSM), that embed variable length sequences in a continuous high-dimension embedding space using a list of short learned kernel sequences of same dimension. Each scalar entry of the embedding vector is calculated using the input sequence and one kernel sequence through the eponymous function (CSM function). The parameters in the kernel sequences are trained in a supervised fashion through a Siamese network structure. Alignment distance between pair of nucleotide sequences are used as continuous valued label. The cosine distance between the pair of embedding vectors is the estimation of the label.

The CSM function works similarly as 1-D Convolutional Neural Network (CNN). The novelty of our approach is in the behavior of the kernels. In order to adapt the insertion/deletion abundance characteristics of biological sequences. We adopted the idea in [15, 16] in our model design. The Kernel sequences interact with the input sequence through alignment in stead of convolution to capture edit invariant features. Further, inspired by the idea of approximate string matching [17, 18], our kernel sequences efficiently scan the input sequence using dynamic programming in stead of fix width sliding window in CNN. Features are extracted from best approximate subsequences (BAS) of variable length rather than subsequence in a fixed width sliding window. Figure 1 presents the overview of the proposed method and its training process. Overall, we adapt the classic Siamese network structure to learn the proposed embedding model. Each kernel scans the input sequence and aligns itself to the best approximate subsequence. The gradients with respect to the kernel sequences are calculated through back-propagation. At last we introduce a scheme to train the model efficiently in batches to stabilize the training process and reduce the convergence time by a factor of 30 compared with training using one pair at a time.

To be self contained we first briefly introduce the definition of alignment score and alignment distance. Then describe the detail of our model.

2.1 Alignment Score

In classic alignment algorithm such as Needleman-Wunsch algorithm and Smith-Waterman algorithm. The alignment is converted to a optimal path search problem on a graph. This problem can be solved using dynamic programming. The optimal path is computed by maximizing an objective function through filling a matrix step by step. The objective function is a summation of user defined gap penalties and substitution scores correspond to each step. The alignment score between a pair of sequences $X : x_1, ..., x_{l_1}$ and $Y : y_1, ..., y_{l_2}$ is defined as the maximum objective function value with gap cost g. In most case stored in the last row last column of the $(l_1 + 1) \times (l_2 + 1)$ matrix F used in the dynamic programming. F is calculated in recursion:

$$F_{i,j} = \begin{cases} 0 & \text{if } i = 0 \\ -j \cdot g & \text{if } j = 0 \\ max(F_{i-1,j-1} + \text{match}(x_i, y_j), F_{i-1,j} - g, F_{i,j-1} - g) & \text{otherwise} \end{cases} \quad (1)$$

The alignment score is noted as $S_{\text{NW}}(X, Y, g) = F[l_1, l_2]$.

2.2 Alignment Distance

Given an alignment between a pair of sequences. The alignment distance is computed as the ratio between number of not matched cites and the length of alignment result. The alignment distance between two sequences X, Y is noted as $d(X, Y, g)$.

2.3 Siamese Network

Our model is designed for nucleotide sequences. The nucleotide sequence is first converted into one-hot encoded sequence for neural network to work on as input. Each entry of the sequence is a 4 dimension vector correspond to the 4 different nucleotides: A, T, G, C. There is exactly one entry of the vector equal to 1 and all other entries are set to 0. The model is used to generating alignment distance preserving embedding vectors for nucleotide sequences.

There are two hyper parameters in our model. The first is the number of feature channels d which is equal to the embedding dimension. Each channel consists of a kernel sequence K_i and a scalar bias b_i where $i \in [1, d]$. The second is the length of kernel sequences l. Kernel sequences of all channel share the same dimension and length. The trainable parameter set of the model consists of d kernel sequences and the gap cost and bias for each kernel is represented as $\omega : \{K_1, ..., K_d, g_1, ..., g_d, b_1, ..., b_d, \}$.

In the single pair case shown in Figure 1 (a), the model takes a pair of one-hot encoded sequences as input note as X_1 and X_2. The embedding vectors for both sequences are calculated using the same set of parameters ω, noted as $\vec{e}(X_1|\omega)$ and $\vec{e}(X_2|\omega)$. The i-th entry of the embedding vector is computed as following through the CSM function which will be introduced in the next subsection:

$$\vec{e}(X|\omega)_i = ReLu(CSM(X, K_i, g_i) + b) \quad (2)$$

where b is a bias term and ReLU is the rectified linear activation function [19]. The model then compute the cosine distance between the embedding vectors as its estimation of the alignment distance. The cosine distance between vector $\vec{u}$ and $\vec{v}$ is defined as the complement of cosine similarity:

$$d_C(u, v) = 1 - \frac{\vec{u}}{\|\vec{u}\|} \cdot \frac{\vec{v}}{\|\vec{v}\|} \quad (3)$$

Then model is trained using gradient descent algorithm to minimize the mean square error given by:

$$\mathscr{L}(\omega) = (d(X_1, X_2, g_i) - d_C(\vec{e}(X_1|\omega), \vec{e}(X_2|\omega)))^2 \quad (4)$$

In the training process, the kernel sequences are random initialized each entry is set to a value between 0 and 1. Once the model converged, it can be used for fast and accurate homologous sequence comparison. In our application the converged kernel sequences can be viewed as sequence motifs since each row correspond to 1 nucleotide based on the one-hot encoded input. In the situations when each character are not considered equal such as amino acids, character embedding vectors are used instead of one-hot vectors, each column of the learned kernel sequences can still be used to infer the extracted pattern.

2.4 Continuous Sequence Matching Function

The CSM function is a modified local-global alignment algorithm using dynamic programming. Given a input high dimension sequence $X : x_1, ..., x_L$ and a kernel sequence $K : k_1, ..., k_n$ ($n \ll L$) of same dimension and a gap penalty g as input. the CSM function find the BAS X' of X that has maximum alignment score aligned with the kernel sequence K among all the sub-sequences of S. Let the start and end index of the BAS be i_s and i_e and the length noted as $m = j_e - j_s + 1$, then the BAS is $X' = x_{i_s}, ..., x_{i_e} = x'_1, ..., x'_m$. The CSM function can be written as:

$$CSM(X, K, g) = \max_{i_s, i_e} \left(S_{\text{NW}}(X', K, g)\right) \tag{5}$$

An example is shown in Figure 1 (b).

In our case the substitution score of the i-th vector and j-th vector is their dot product. An commonly used algorithms [18] identifies the best matching sub-sequence and calculates the best matching score using dynamic programming. In particular, it constructs a $(m+1) \times (n+1)$ matrix T and fill it recursively as following:

$$T_{i,j} = \begin{cases} 0 & \text{if } i = 0 \\ -j \cdot g & \text{if } j = 0 \\ max(T_{i-1,j-1} + x_i \cdot k_j, T_{i-1,j} - g, T_{i,j-1} - g) & \text{otherwise} \end{cases} \tag{6}$$

The (i, j) entry $T_{i,j}$ represents the maximum alignment score between any sub-sequence $X_{i':i} = x_{i'} ... x_i$, $i' < i$ of X that ends at position i and the prefix of the kernel sequences of length j, $K : k_1 ... k_j$. The maximum value of the last row of T stores the maximum alignment score between any sub-sequence ends at any index and the full kernel sequence, which is the output of the CSM function. Note that the length of the BAS is not always equal to the length of the kernel sequence since the starting index of the it is calculated by the standard backtracking algorithm [18]. The CSM function can calculate the output value without backtracking the start index i_s. This reduce the running time in testing process. However the start index is required in the backpropagation stage. An example of alignment between K and one-hot encoded DNA sub-sequence is shown in Figure 1 (c). The left sub-sequence is aligned with the kernel with 1 gap and have a CSM function output 5.7 and the right one is aligned with 2 gaps and have CSM function output 4.6.

As the gap penalty increases to infinity. The embedding method is degenerated to a 1-D single layer CNN with stride of 1 and no padding followed by a global max-pooling layer. As $g \to \infty$, the matrix F is calculated as:

$$T_{i,j} = T_{i-1,j-1} + x_i \cdot k_j, (i \cdot j \neq 0) \tag{7}$$

The CSM function allows kernel sequences to extract more robust feature from the input in the presence of insertion/deletion which increases the generalization ability of the model. In the next section, a perturbation experiment will show the advantage of CSM over CNN.

2.5 Gradient Computation version

In the training process, gradient of the loss function with respect to (w.r.t) each parameter is used to update our model by the well known back-propagation algorithm. Based on the chain rule, the gradient can be written as follows

$$\frac{\partial \mathscr{L}(\omega)}{\partial \omega} = \left[\frac{\partial \mathscr{L}(\omega)}{\partial CSM(X, K, g)} \frac{\partial CSM(X, K, g)}{\partial K}, \frac{\partial \mathscr{L}(\omega)}{\partial b} \right] \tag{8}$$

The gradient of elementary functions can be calculated automatically by standard machine learning library like Pytorch and Tensorflow. Then we only need to discuss the gradient computation of the CSM function w.r.t the kernel sequences. In fact, only the entries on the optimal alignment path between kernel sequence and the BAS contribute to the output value of CSM function. However, the optimal alignment is settled down only in the current iteration. The alignment will possibly change in the next iteration which result in high variance in the training process. In our implementation we applied the soft version of Needleman-Wunsch algorithm[15] to approximate the gradient. The soft-max function

is used to replace the discontinuous max function in the recursion to make the dynamic processing differentiable.

$$max^{\gamma}(\vec{v}) = \gamma \log\left(\sum_i \exp\left(v_i/\gamma\right)\right) \tag{9}$$

$$\frac{\partial max^{\gamma}(\vec{v})}{\partial v_i} = exp(\frac{v_i - max^{\gamma}(\vec{v})}{\gamma}) \tag{10}$$

The soft alignment score between the BAS and kernel is noted as $S^{\gamma}_{\mathrm{NW}}(X', K, g)$. The hyperparameter γ controls the trade-off between accuracy and smoothness. But in our case, the direct application of this soft approach will result in $O(lL)$ time complexity, again, L is the length of the input sequence due to the soft-max function unnecessarily blending the possible alignment outside the BAS into the gradient computation which hardly has contribution to the output. To reduce the computation burden we only use the BAS to compute the gradient.

$$\frac{\partial CSM(X,K,g)}{\partial K} \approx \frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial K} \tag{11}$$

The recursion in the computation of F changes to

$$F_{i,j} = \begin{cases} -i \cdot g & \text{if } i = 0 \\ -j \cdot g & \text{if } j = 0 \\ max^{\gamma}(F_{i-1,j-1} + x'_i \cdot k_j, F_{i-1,j} - g, F_{i,j-1} - g) & \text{otherwise} \end{cases} \tag{12}$$

The gradient w.r.t the kernel and gap cost can be written as:

$$\frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial k_j} = \sum_{i=1}^{i_e - i_s + 1} \frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i,j}} \frac{\partial F_{i,j}}{\partial k_j} \tag{13}$$

$$\frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial g} = \frac{\partial F_{m,n}}{\partial g} \tag{14}$$

We can see, the calculation of $\frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i,j}}$ can be done recursively with stop condition $\frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i_e - i_s + 1, m}} = 1$. The recursive call is derived from the fact that each entry $F_{i,j}$ only contribute to the three neighbors $F_{i+1,j}, F_{i+1,j+1}, F_{i,j+1}$. This term can be write as:

$$\frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i,j}} = \frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i+1,j}} \cdot \frac{\partial F_{i+1,j}}{\partial (F_{i,j} - g)} + \frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i+1,j+1}} \cdot \frac{\partial F_{i+1,j+1}}{\partial (F_{i,j} + x'_{i+1} \cdot k_{j+1})} + \frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i,j+1}} \cdot \frac{\partial F_{i,j+1}}{\partial (F_{i,j} - g)} \tag{15}$$

The gradient w.r.t the gap cost g is also computed recursively, with stop condition: $\frac{\partial F_{i,0}}{\partial g} = -i$ and $\frac{\partial F_{0,j}}{\partial g} = -j$:

$$\frac{\partial F_{i,j}}{\partial g} = \frac{\partial F_{i,j}}{\partial (F_{i-1,j} - g)} \cdot \frac{\partial (F_{i-1,j} - g)}{\partial g} + \frac{\partial F_{i,j}}{\partial (F_{i-1,j-1} + x'_i \cdot k_j)} \cdot \frac{\partial (F_{i-1,j-1} + x'_i \cdot k_j)}{\partial g} + \frac{\partial F_{i,j}}{\partial (F_{i,j-1} - g)} \cdot \frac{\partial (F_{i,j-1} - g)}{\partial g} \tag{16}$$

For simplicity purpose, we write $\frac{\partial S^{\gamma}_{\mathrm{NW}}(X',K,g)}{\partial F_{i,j}}$ as $Q_{i,j}$ and $\frac{\partial F_{i,j}}{\partial g}$ as $P_{i,j}$. The approximate gradient computation of the CSM function is given in Algorithm 1:

Algorithm 1: Gradient computation CSM function

Input: Sequence: $X : x_1,...,x_L$, kernel: $K : k_1,...,k_m$, gap cost: g, soft-max hyperparameter: γ

1 Initialize matrix $T = (L+1) \times (n+1)$ with $T_{i,0} = -ig, \forall i \in [1,m]$ and fill the matrix T for all $i \cdot j \neq 0$:

$$T_{i,j} = max(T_{i-1,j-1} + x_i \cdot k_j, T_{i-1,j} - g, T_{i,j-1} - g)$$

2 Compute the BAS region: $i_e = \arg\max_i T_{i,n}$, trace back from j_e to $j = 0$ to get i_s. BAS length: $n = j_e - j_s + 1$, retrieve $X' = x_{j_s},...,x_{j_e}$ note as $x'_1,...,x'_n$

3 Initialize zero matrix $F = (m+2) \times (n+2)$ with $F_{i,0} = -ig$, $F_{i,n+1} = \infty, \forall i \in [0,m]$, $F_{0,j} = -jg$, $F_{m+1,j} = \infty, \forall j \in [1,n]$. Fill F for $\forall (i,j) \in [1,m] \times [1,n]$:

$$F_{i,j} = max^{\gamma}(F_{i-1,j-1} + x'_i \cdot k_j, F_{i-1,j} - g, F_{i,j-1} - g)$$
$$F_{m+1,n+1} = F_{m,n}$$

4 Initialize zero matrix $Q = (m+2) \times (n+2)$ with $Q_{m+1,n+1} = 1$, and fill Q for $\forall (i,j) \in [1,m] \times [1,n]$:

$$Q_{i,j} = Q_{i+1,j} \cdot \exp(\tfrac{F_{i,j} - g - F_{i+1,j}}{\gamma}) + Q_{i+1,j+1} \cdot \exp(\tfrac{F_{i,j} + x'_{i+1} \cdot k_{j+1} - F_{i+1,j+1}}{\gamma}) + Q_{i,j+1} \cdot \exp(\tfrac{F_{i,j} - g - F_{i,j+1}}{\gamma})$$

5 Compute the gradient w.r.t kernel ∇K:

$$\nabla K_j = \sum_{i=1}^{i_e - i_s + 1} Q_{i,j} \exp\left(F^{\gamma}_{i-1,j-1} + x'_i k_j - F^{\gamma}_{i,j}/\gamma\right) x'_i$$

6 Initialize zero matrix $P = (m+2) \times (n+2)$ with $P_{i,0} = -i$, $\forall i \in [0,m]$, $P_{0,j} = -j$, $\forall j \in [1,n]$ and fill the matrix P for $\forall (i,j) \in [1,m] \times [1,n]$:

$$P_{i,j} = \exp(\tfrac{F_{i-1,j} - g - F_{i,j}}{\gamma}) \cdot (P_{i-1,j} - 1) + \exp(\tfrac{F_{i-1,j-1} + x'_i \cdot k_j - F_{i,j}}{\gamma}) \cdot P_{i-1,j-1} + \exp(\tfrac{F_{i-1,j} - g - F_{i,j}}{\gamma}) \cdot (P_{i,j-1} - 1)$$

$$\nabla g = P_{m,n}$$

Return: $\nabla K, \nabla g$

2.6 Fast Random Batch

In the training process, the computation of embedding vectors using the CSM function is the bottle neck. Our default model have 200 kernel sequences of length 20. Without parallel computing, the embedding process is equivalent to do alignment between the input sequence of length approximately 1500 and parameter sequence of length 4000. This process is required for both sequences fed to the model in each update makes the training for a pair even slower than the alignment. The high variance in the training process results in slow convergence. Therefore requiring more update steps. Collectively make the training process time consuming.

We propose a method to train the model efficiently by random sampled batches from a large pairwise training matrix. In each iteration, n sequences are randomly sampled and the $(n^2 - n)/2$ pairwise labels are extracted from the training set forms a small alignment distance matrix D. The model compute the embedding vector for the sampled sequences using current parameters. Then a pairwise cosine distance matrix of the embedding vectors D_C is computed as prediction. The mean squared error between the label matrix and the predicted matrix is minimized.

$$\mathscr{L}(\omega) = \|D(S) - D_C(\vec{e}(S|\omega))\|_F \tag{17}$$

Although batch training stabilizes the gradient descent process, it does not guarantee reduction of training time since there is a trade-off between update frequency and update effectiveness. If the batch size is too large, even the model converge after fewer updates, it is possible each updates cost too much time that the training process end up slower than training without batch. For our method, in each iteration, the embedding vector is calculated $O(n)$ times and the gradient is averaged among $O(n^2)$ prediction-label pairs. This allow us to have enough distance label to smooth our

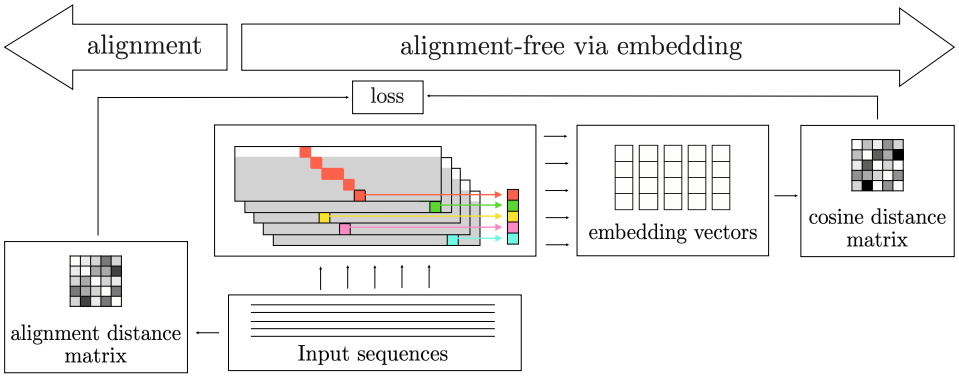


Figure 2: Overview of the network structure used in batch training.

gradient while the time cost of each iteration dominated by the CSM function is still small enough to be collectively time conserving. Based on our experiment, $n = 5$ is a good choice.

2.7 Related Work

We compared the proposed method with 3 data-independent and 2 data-dependent methods. For data-independent methods, we compared with kmacs [13], k-mer [12], FFP [20] since they had shown competitive performance in previous study [14]. See [11] for a complete review of alignment free methods. For data-dependent methods, the first method is SENSE [14], which is the first data dependent method for sequence comparison. It uses a Siamese neural network to learn a CNN-based embedding function based on training datasets. The second method, referred to as single-layer CNN, is a gap-free version of the proposed method. In particular, if we do not allow insertion/deletion in the proposed CSM function, it will degenerate to a single-layer CNN followed by a global maximum pooling function. We compare our method with single-layer CNN to assess the effectiveness of introducing insertion/deletion.

3 Experimental Results

To demonstrate the effectiveness and efficiency of the proposed method, we performed large-scale experiments on four real-world sequence datasets and compare the performance with five competing methods.

3.1 Dataset

Four amplicon sequence datasets were used in our experiment. We used two full-length 16S rRNA datasets. The first dataset is the Greengenes full-length 16S rRNA database [21], which contains $1,262,986$ curated sequences of length ranging from $1,111$ to $2,368$. The second dataset was generated from a Zymo mock community [5]. The samples in this dataset were sequenced by an amplicon sequencing methodology based on PacBio CCS sequencing, which can produce accurate full-length 16S rRNA sequences without assembly. As CCS reads are generated in a mixture of forward and reverse-complement orientations, we performed a pre-processing procedure using the remove Primers function in the DADA2 R package [22] to remove primers and orients all the sequences in the forward direction. After pre-processing, the Zymo dataset contains $69,367$ sequences of length ranging from $1,187$ to $1,518$. We also use two datasets that covers the subregions of the 16S rRNA gene. Qiita was generated from 66 skin, saliva and feces samples collected from Yanomani, the uncontacted Amerindians [23]. It contains $6,734,572$ sequences of 151 bp that cover the V4 hyper-variable region of the 16S rRNA gene. RT988 contains $4,119,942$ sequences from 90 oral plaque samples, each of which is 464-465 bp in length and covers the V3-V4 regions. For both Qiita and RT988, we performed pre-processing procedures including pair-end joining, quality filtering and length filtering before the analysis. For all the datasets, we removed duplicate sequences.

3.2 Experimental Setting

3.2.1 Training and Testing Data

All data-dependent methods (CSM, single-layer CNN and SENSE) require training samples to tune parameters. For our method and the single-layer CNN, training sequences were generated by randomly selecting 5 groups of 1000 sequences from each of Greengenes, Qiita and RT988. For SENSE, we randomly selected sequences and trimmed them to fixed lengths (Qiita: 151, RT988: 465) since it can only take fix-length sequences as input. We also randomly sampled 10 groups of 1000 sequences from Qiita, RT988, Greengenes and Zymo in order to compare the performance. Again, the testing sequences for SENSE were trimmed to fixed length (Qiita: 151, RT988: 465). For each group of sequences, we generated the pairwise alignment of all $499,500$ possible sequence pairs by using the NW algorithm implemented in seqAn [24] with NCBI default parameters (match score: 2, mismatch score: -3, gap existence cost: -5, gap extension cost: -2) and calculated the alignment distances the same way as in [14]. Batch size is set to 5.

3.2.2 Hyperparameter Selection

For our method, we used 200 kernel sequences of length 20 across all the experiments. To train our model, we used the Adam optimizer [25] to minimize the mean square error using stochastic gradient descent with the learning rate 10^{-3}. For single-layer CNN, we used the exactly same setting as our method except that it does not have the gap parameter. For SENSE, the number of CNN layers is set to 3, filter length to 11, the number of filters in three layers to 16, 32 and 48, and the length of the output embedding vectors is the same as the length of the input sequence. For kmacs and FFP (V3.19), we used the source code downloaded from their websites. For k-mer, we used our own C++ implementation optimized for amplicon sequence data by using sparse k-mer count representation. For k-mer, kmacs and FFP, we tested 10 parameters for each method since there is no principal way to estimate the optimal parameter.

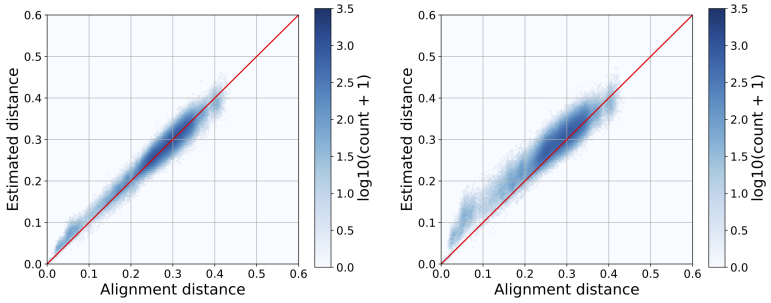
3.3 Average Relative Error as Accuracy Measurement

In this paper, we used the mean relative error (MRE) to measure the effectiveness of the algorithms. The relative error is calculated by dividing the absolute difference/error between the predicted distance and the aligned distance by the aligned distance. Due to the large range of the alignment distance (from < 0.01 to > 0.3), relative error is more suitable for measuring the quality of approximation than the absolute error. Furthermore, the physical meaning of relative error is more clear. For example, if the MRE of an algorithm is 10%, it means that on average, the prediction of this algorithm deviates by 10% from the true alignment distance, regardless of the value of the true alignment distances.

3.4 Perturbation Experiment

The purpose of the perturbation experiment is to justify the necessity of gap awareness. The sequences sampled from large curated database are usually from different species. Therefore their 16S rRNA gene sequence similarity $< 98.7\%$ [26]. Difference between two sequences of same species can be viewed as perturbation. This experiment shows the CSM model can generate more robust embedding vectors in the presence of individual differences which is crucial in many sequence analysis application. The experiment is performed on synthetic data to simulate slight individual differences. First, 200 sequences are randomly sampled from the Greengene database. Then for each sequence, 5 slightly different synthetic sequences are generated by randomly insert, delete and substitute nucleotide at each position with probability 0.01. The pairwise alignment distance of the 1000 generated sequences are then used for testing. The result is shown in Figure 3.

We can see the CNN model tend to give larger prediction, especially on pairs with smaller alignment distance. Miss detection of similar pairs can be problematic in many sequence based analysis. For example, when doing database search, the query sequence is not 100% identical to the sequence of the correct species, CSM can generate more similar embedding vectors for slightly different sequences compare to CNN.

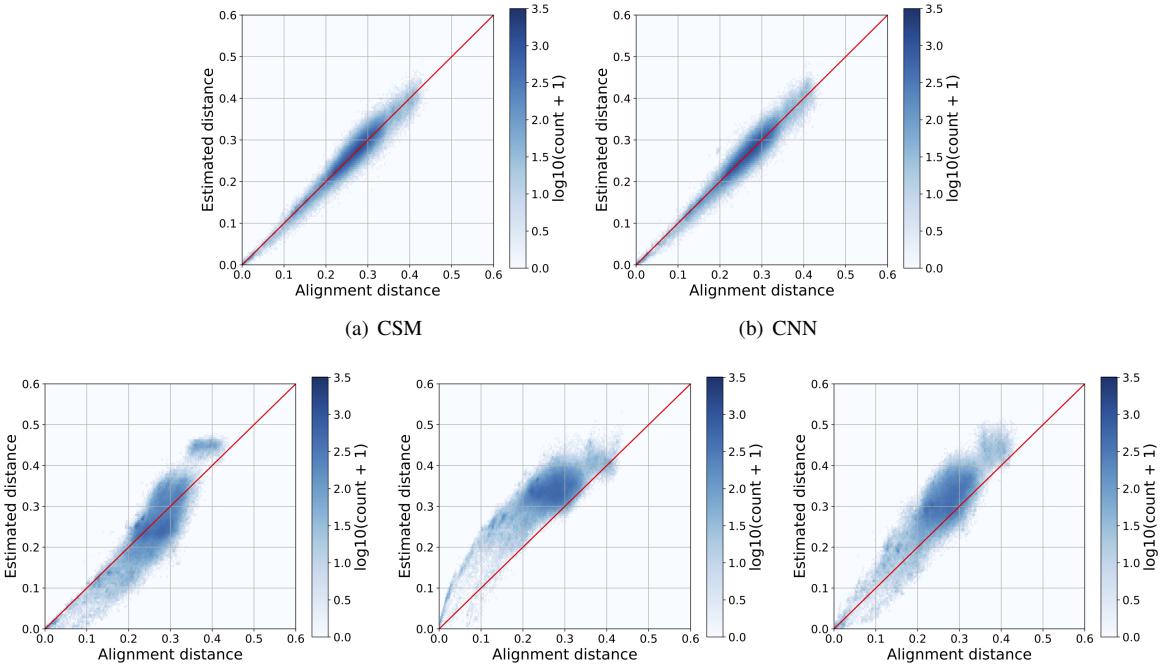


(a) CSM (b) CNN

Figure 3: Perturb analysis. (a) Result of CSM model, MRE 4.5%. (b) Result of CSM model, MRE 9.5%.

3.5 Benchmark Analysis

First we compare the predictive abilities of all the methods on the full-length 16S sequences. Two non overlap sets are extracted from the Greengenes database, training and testing. The three data-dependent methods, are trained on the training set. All methods are tested on testing sets and Zymo dataset to compare the generalization ability of the CSM model and its CNN special case. Table 1 reports the MRE and CPU time on these two datasets and Figure 4 and Figure 5 plot the estimated distances against the alignment distance for the Greengenes and Zymo, respectively. On Greengenes, we can see that CSM and single-layer CNN performed best with a large margin. SENSE and kmacs are distant second with kmacs. Considering that SENSE is trained on Greengene, we can conclude that SENSE failed to capture the characteristics of the full-length 16S sequences, even if it has a much more number of parameters to adjust. In contrast, the CSM model used much less parameters (4200) to achieve a higher accuracy.



(a) CSM (b) CNN

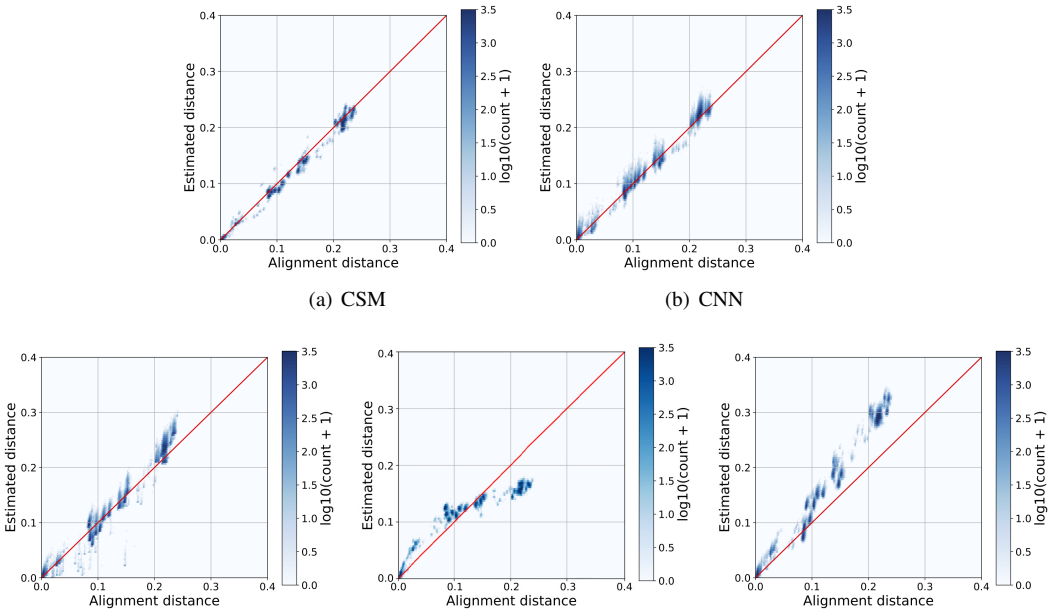
(c) kmacs (k=2) (d) k-mer (k=5) (e) FFP (l=12)

Figure 4: Visualization of alignment distances versus estimated distances computed by six methods performed on Greengenes dataset.

The results on Zymo has an obvious feature that the accuracy of all methods has decreased, whether they are data-dependent or data-independent. One possible explanation is that, compared to sequences in the Greengene database, the sequences in the Zymo dataset are raw sequences that have more variations, which tends to make the alignment distances more difficult to predict. Remarkably, despite this difficulty, CSM still performed best among all methods. It is also worth noting that single-layer CNN can no longer match the performance with CSM. The implication of these results is twofold. First, this result shows that introducing insertion/deletion provides a strong resistance to the variation in the data. Second, it shows that the proposed method is able to build a general model for full-length 16S rRNA sequences. Once a model is trained, we can applied it to predict the alignment distance between any pair of full-length sequences with a high accuracy guarantee, no matter where the sequences come from and whether they are curated. Such model can greatly facilitate the applications based on 16S sequence comparison due to its effectiveness and efficiency.

		Greengenes			Zymo		
Method	**Parameter**	**CPU time (s)**	**MRE**	***p*-value**	**CPU time (s)**	**MRE**	***p*-value**
CSM	default	40.5 (0.2)	**4.5% (0.01%)**	–	41.2 (0.05)	**13.1% (0.4%)**	–
Single-layer CNN	default	24.3 (1.7)	**4.8% (0.06%)**	3.5e-08	24.0 (0.5)	**20.9% (1.1%)**	5.3e-04
kmacs	$k=1$	189.1 (0.9)	16.8% (0.3 %)		191.3 (0.8)	36.4% (0.7%)	
	$k=2$	210.1 (1.7)	**12.7% (0.2%)**	1.6e-29	214.7 (1.0)	**24.4% (0.4%)**	3.1e-06
	$k=3$	229.7 (1.0)	18.0% (0.3%)		231.9 (0.8)	28.3% (0.3%)	
	$k=4$	249.1 (2.8)	25.4% (0.3%)		248.2 (0.3)	37.6% (0.4%)	
	$k=5$	269.7 (4.7)	31.8% (0.3%)		263.6 (0.3)	44.1% (0.4%)	
	$k=6$	294.3 (9.6)	36.8% (0.3%)		280.7 (0.8)	48.3% (0.4%)	
	$k=7$	303.2 (1.9)	41.0% (0.3%)		293.0 (0.5)	51.9% (0.4%)	
	$k=8$	317.3 (2.1)	44.3% (0.3%)		304.4 (0.2))	55.0% (0.4%)	
	$k=9$	342.6 (16.0)	47.2% (0.3%)		316.8 (0.6)	57.6% (0.4%)	
	$k=10$	362.6 (25.1)	49.6% (0.2%)		327.4 (0.3)	59.7% (0.4%)	
***k*-mer**	$k=3$	3.4 (0.01)	67.7% (0.3%)		3.5 (0.02)	53.7% (0.4%)	
	$k=4$	9.9 (0.05)	35.7% (0.3%)		10.0 (0.4)	**40.1% (0.4%)**	1.2e-07
	$k=5$	26.7 (0.2)	**30.1% (0.5%)**	8.5e-29	26.8 (0.3)	107.0% (1.4%)	
	$k=6$	44.0 (0.1)	103.5% (0.7%)		43.3 (0.1)	200.7% (1.7%)	
	$k=7$	51.2 (0.7)	154.2% (0.8%)		51.0 (0.1)	271.7% (1.9%)	
	$k=8$	52.6 (0.5)	181.0% (0.8%)		53.0 (0.2)	318.9% (2.4%)	
	$k=9$	52.3 (0.3)	181.1% (0.8%)		53.0 (0.2)	318.7% (2.4%)	
	$k=10$	52.1 (0.1)	181.1% (0.8%)		53.0 (0.2)	318.7% (2.4%)	
	$k=11$	52.0 (0.01)	181.1% (0.8%)		53.1 (0.3)	318.6% (2.4%)	
	$k=12$	52.1 (0.1)	181.2% (0.8%)		53.1 (0.3)	318.8% (2.4%)	
FFP	$l=6$	3.2 (0.02)	98.1% (0.05%)		3.1 (0.3)	98.0% (0.0%)	
	$l=7$	5.2 (0.2)	96.0% (0.05%)		5.1 (0.4)	95.6% (0.0%)	
	$l=8$	11.0 (0.3)	91.6% (0.1%)		10.4 (0.8)	90.3% (0.0%)	
	$l=9$	19.9 (0.7)	84.7% (0.1%)		19.2 (1.6)	80.9% (0.1%)	
	$l=10$	40.7 (1.0)	93.9% (0.1%)		39.1 (3.0)	86.9% (0.1%)	
	$l=11$	76.1 (0.7)	56.4% (0.2%)		74.1 (6.4)	57.0% (0.3%)	
	$l=12$	149.2 (0.8)	**18.0% (0.3%)**	2.6e-29	143.1 (11.8)	**42.8% (0.2%)**	1.7e-08
	$l=13$	274.1 (0.9)	77.3% (0.4%)		250.0 (12.1)	128.7% (0.7%)	
	$l=14$	527.2 (2.1)	120.0% (0.5%)		382.4 (2.5)	199.1% (0.6%)	
	$l=15$	1031.8 (3.5)	147.1% (0.6%)		548.5 (15.4)	274.2% (3.0%)	

Table 1: CPU time and MRE results averaged over ten runs for six methods performed on the Greengenes and Zymo datasets. Standard deviations are shown in bracket. The best result for each method is boldfaced. A p-value was computed by comparing the MRE result of CSM with the best result of each method.

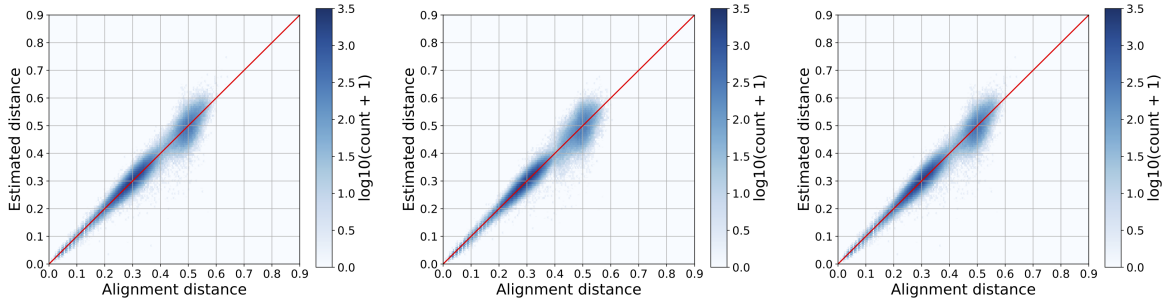


(a) CSM (b) CNN

(c) kmacs (k=2) (d) *k*-mer (k=4) (e) FFP (l=12)

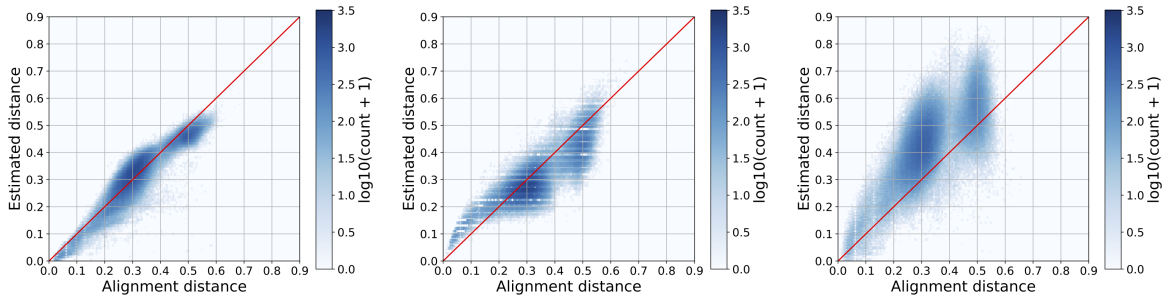
Figure 5: Visualization of alignment distances versus estimated distances computed by six methods performed on Zymo dataset.

To demonstrate that the ability of proposed method is not limited to full-length 16S rRNA sequences, we also performed similar analysis on Qiita and RT988 dataset. The CPU times and MRE in Supplementary Table 2, and the Figures 6 and Figure 7. We can see that the three data-dependent methods are much better than other methods in terms of accuracy and are at least as fast as the methods other than k-mer. Among them, CSM and single-layer CNN performed slightly better than SENSE on both datasets with fewer parameters (CSM and single-layer CNN: 4200, SENSE: $>$ 30000).



(a) CSM (b) CNN (c) SENSE

Figure 6: Visualization of alignment distances versus estimated distances computed by six methods performed on Qiita dataset.

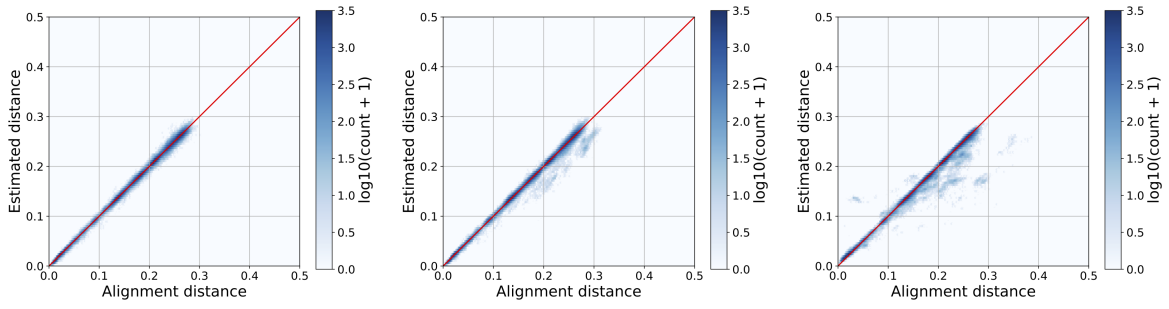


(a) kmacs (k=2) (b) *k*-mer (k=4) (c) FFP (l=12)

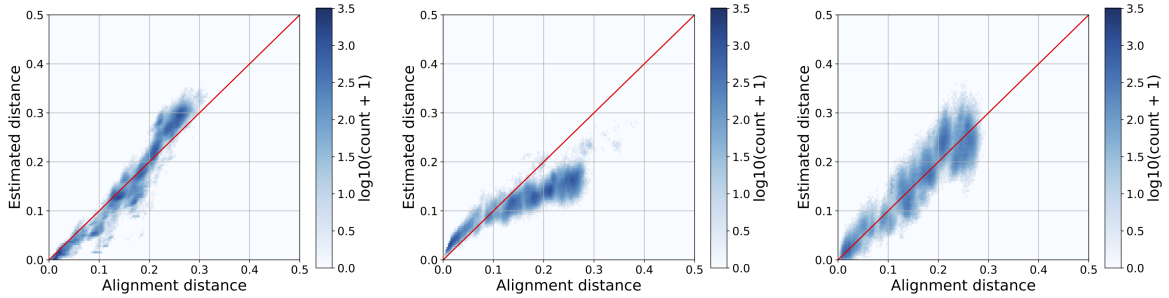
Figure 6: Visualization of alignment distances versus estimated distances computed by six methods performed on Qiita dataset.

Table 2: CPU time (in second) and MSE results averaged over ten runs for six methods performed on the Qiita and RT988 datasets. The numbers in parentheses are standard deviations. The best result is boldfaced for kmacs, k-mer and FFP. A *p*-value was computed by comparing the MSE result of CSM with the best results of other methods.

		Qiita			RT988		
Method	**Parameter**	**CPU time (s)**	**MRE**	***p*-value**	**CPU time (s)**	**MRE**	***p*-value**
CSM	default	17.6 (0.6)	**4.6% (0.1%)**	–	23.4 (0.2)	**2.6% (0.1%)**	–
Single-layer CNN	default	5.4 (0.1)	**4.4% (0.1%)**	5.0e-05	9.6 (0.6)	**2.7% (0.1%)**	1.8e-01
SENSE	default	9.7 (0.3)	**6.6% (0.1%)**	5.2e-21	18.6 (0.1)	**3.9% (0.2%)**	8.0e-12
kmacs	$k=1$	22.1 (0.6)	21.2% (0.3%)		62.7 (1.5)	27.2% (0.2%)	
	$k=2$	24.2 (0.5)	**10.2% (0.1%)**	5.8e-27	69.8 (1.2)	**16.4% (0.1%)**	1.8e-34
	$k=3$	25.8 (0.2)	14.4% (0.2%)		76.5 (1.6)	21.6% (0.4%)	
	$k=4$	27.6 (0.2)	23.7% (0.2%)		82.5 (1.7)	31.2% (0.5%)	
	$k=5$	29.5 (0.4)	31.2% (0.2%)		88.2 (2.5)	39.0% (0.6%)	
	$k=6$	31.6 (0.4)	37.0% (0.2%)		93.4 (2.7)	45.0% (0.5%)	
	$k=7$	33.1 (0.9)	41.7% (0.2%)		96.5 (1.7)	50.1% (0.5%)	
	$k=8$	34.5 (0.8)	45.7% (0.2%)		100.0 (1.3)	54.1% (0.5%)	
	$k=9$	35.9 (1.0)	49.2% (0.2%)		104.4 (2.2)	57.1% (0.4%)	
	$k=10$	37.0 (0.8)	52.2% (0.2%)		108.1 (2.9)	59.6% (0.4%)	
***k*-mer**	$k=3$	3.1 (0.1)	**14.9% (0.3%)**	3.1e-26	3.3 (0.1)	36.8% (0.6%)	
	$k=4$	5.3 (0.2)	65.8% (0.6%)		8.3 (0.3)	**62.7% (1.6%)**	2.6e-27
	$k=5$	6.4 (0.3)	129.5% (1.1%)		14.0 (0.1)	147.8% (1.5%)	
	$k=6$	6.7 (0.3)	167.3% (1.4%)		16.9 (0.1)	219.1% (1.8%)	
	$k=7$	6.6 (0.2)	188.4% (1.7%)		17.9 (0.2)	267.6% (2.2%)	
	$k=8$	6.5(0.2)	200.7% (1.9%)		18.0 (0.1)	301.4% (2.7%)	
	$k=9$	6.5 (0.2)	201.2% (2.0%)		18.0 (0.1)	302.1% (2.7%)	
	$k=10$	6.5 (0.2)	201.4% (2.0%)		17.9 (0.1)	301.0% (2.6%)	
	$k=11$	6.5 (0.2)	201.4% (2.0%)		17.9 (0.1)	300.0% (2.6%)	
	$k=12$	6.4 (0.2)	201.4% (2.0%)		17.9 (0.1)	299.0% (2.5%)	
FFP	$l=6$	2.9 (0.1)	87.9% (0.2%)		2.9 (0.01)	93.0% (0.1%)	
	$l=7$	5.0 (0.1)	90.0% (0.1%)		4.9 (0.05)	85.4% (0.1%)	
	$l=8$	9.8 (0.2)	38.7% (0.3%)		9.9 (0.02)	83.2% (0.1%)	
	$l=9$	17.4 (0.7)	**38.5% (0.5%)**	4.3e-32	18.0 (0.05)	78.0% (0.2%)	
	$l=10$	33.2 (1.1)	97.4% (1.0%)		35.5 (0.04)	**18.4% (0.5%)**	3.1e-25
	$l=11$	61.2 (2.4)	137.5% (1.4%)		64.9(0.05)	95.6% (1.1%)	
	$l=12$	121.6 (4.5)	162.8% (1.8%)		123.7 (0.4)	169.8% (1.7%)	
	$l=13$	233.1 (6.4)	178.9% (2.0%)		219.8 (1.0)	220.1% (2.2%)	
	$l=14$	428.3 (5.9)	189.4% (2.1%)		353.4 (3.4)	251.0% (2.6%)	
	$l=15$	692.5 (12.0)	196.4% (2.2%)		493.4 (7.0)	274.2% (3.0%)	

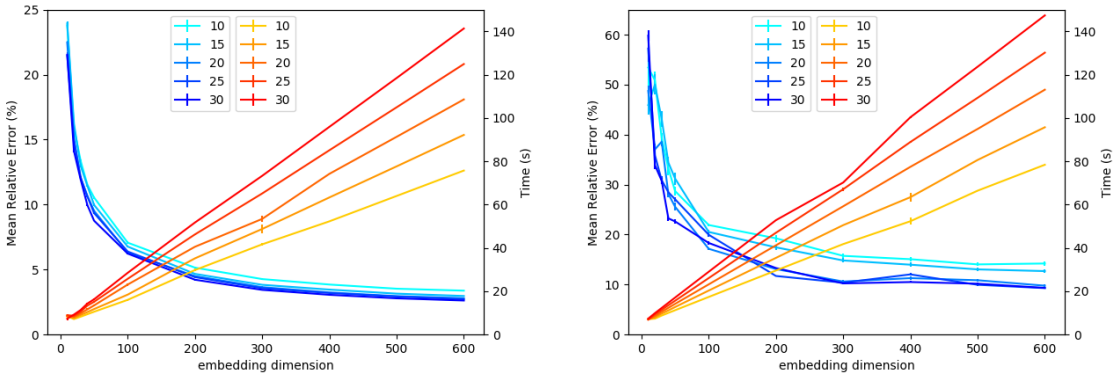


(d) CSM (e) CNN (f) SENSE



(g) kmacs (k=2) (h) *k*-mer (k=4) (i) FFP (l=12)

Figure 7: Visualization of alignment distances versus estimated distances computed by six methods performed on RT988 dataset.



(a) Greengenes (b) Zymo

Figure 8: Investigation of the effect of different parameters on the performance of the proposed model. (a, b) Results of parameter sensitive analysis of gap cost g performed on the Greengenes and Zymo datasets. (c, d) Estimate model complexity by using the elbow method for Greengenes and Zymo datasets.

3.6 Estimating Model Complexity

The only two parameters of our method are the number and the length of the string parameters. Across all the experiments, we simply used 200 string parameters of length 20 and obtained good results. Here we further investigate the effect of these two parameters on the results. As these two parameters determine the complexity of the model, we expected to observed the elbow phenomenon, that is, when the complexity of the model increases, the error quickly

decreases and then flattens at a certain point. To validate this assumption, we trained several models with different combination of parameters with number ranging from 10 to 600 and length ranging from 10 to 30. Figure 8(a) and 8(b) report the MRE of different models applied to Greengenes and Zymo datasets. On Greengenes, we can clearly observed a elbow phenomenon for the number of strings and the models with length ranging from 20 to 30 have similar results. On Zymo, although the values of MRE are larger, we can observed the same trend. Therefore, we choose 300 and 20 as the default parameters.

4 Conclusion

In this paper, we developed a novel method that is specifically designed to capture the characterizes of biological sequences. We demonstrated that the proposed method is able to produce a general model for full-length 16S rRNA amplicon sequences. This model is able to efficiently and accurately predict arbitrary pair of full-length 16S sequences, whether the sequences are curated or directly from study. In addition, our model also performed good on subregions of 16S rRNA gene. In the future, we plan to develop a tool for universally comparison of full-length 16S rRNA genes and further explore the power of the proposed CSM function. One possible direction is to aggregate the CSM function to form complex multiple-layer structures. It would also be interesting to look into the trained function to identify what it learned from data, especially for multiple-layer structures.

References

[1] Ames, S. K., Hysom, D. A., Gardner, S. N., Lloyd, G. S., Gokhale, M. B., and Allen, J. E. (2013) Scalable metagenomic taxonomy classification using a reference genome database. *Bioinformatics,* **29**(18), 2253–2260.

[2] Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T. J., Higgins, D. G., and Thompson, J. D. (2003) Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Research,* **31**(13), 3497–3500.

[3] Li, H. (2016) Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics,* **32**(14), 2103–2110.

[4] Sedlazeck, F. J., Lee, H., Darby, C. A., and Schatz, M. C. (2018) Piercing the dark matter: bioinformatics of long-range sequencing and mapping. *Nature Reviews Genetics,* **19**(6), 329–346.

[5] Callahan, B. J., Wong, J., Heiner, C., Oh, S., Theriot, C. M., Gulati, A. S., McGill, S. K., and Dougherty, M. K. (2019) High-throughput amplicon sequencing of the full-length 16S rRNA gene with single-nucleotide resolution. *Nucleic Acids Research,* **47**(18), e103.

[6] Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., Iyer, R., Schatz, M. C., Sinha, S., and Robinson, G. E. (2015) Big data: astronomical or genomical?. *PLoS Biology,* **13**(7), e1002195.

[7] Roberts, R. J., Carneiro, M. O., and Schatz, M. C. (2013) The advantages of SMRT sequencing. *Genome Biology,* **14**(7), 405.

[8] Jain, M., Olsen, H. E., Paten, B., and Akeson, M. (2016) The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology,* **17**(1), 239.

[9] van Dijk, E. L., Jaszczyszyn, Y., Naquin, D., and Thermes, C. (2018) The third revolution in sequencing technology. *Trends in Genetics,* **34**(9), 666–681.

[10] Needleman, S. B. and Wunsch, C. D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology,* **48**(3), 443–453.

[11] Zielezinski, A., Vinga, S., Almeida, J., and Karlowski, W. M. (2017) Alignment-free sequence comparison: benefits, applications, and tools. *Genome Biology,* **18**(1), 186.

[12] Kariin, S. and Burge, C. (1995) Dinucleotide relative abundance extremes: a genomic signature. *Trends in Genetics,* **11**(7), 283–290.

[13] Leimeister, C.-A. and Morgenstern, B. (2014) Kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison. *Bioinformatics,* **30**(14), 2000–2008.

[14] Zheng, W., Yang, L., Genco, R. J., Wactawski-Wende, J., Buck, M., and Sun, Y. (2019) SENSE: Siamese neural network for sequence embedding and alignment-free comparison. *Bioinformatics,* **35**(11), 1820–1828.

[15] Koide, S., Kawano, K., and Kutsuna, T. (2018) Neural edit operations for biological sequences. In *Advances in Neural Information Processing Systems* pp. 4960–4970.

[16] Cai, X., Xu, T., Yi, J., Huang, J., and Rajasekaran, S. (2019) DTWNet: a dynamic time warping network. In *Advances in Neural Information Processing Systems* pp. 11636–11646.

[17] Sellers, P. H. (1980) The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms,* **1**(4), 359–373.

[18] Ukkonen, E. (1985) Finding approximate patterns in strings. *Journal of Algorithms,* **6**(1), 132–137.

[19] Nair, V. and Hinton, G. E. (2010) Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* pp. 807–814.

[20] Sims, G. E., Jun, S.-R., Wu, G. A., and Kim, S.-H. (2009) Alignment-free genome comparison with feature frequency profiles (FFP) and optimal resolutions. *Proceedings of the National Academy of Sciences,* **106**(8), 2677–2682.

[21] McDonald, D., Price, M. N., Goodrich, J., Nawrocki, E. P., DeSantis, T. Z., Probst, A., Andersen, G. L., Knight, R., and Hugenholtz, P. (2012) An improved Greengenes taxonomy with explicit ranks for ecological and evolutionary analyses of bacteria and archaea. *The ISME Journal,* **6**(3), 610.

[22] Callahan, B. J., McMurdie, P. J., Rosen, M. J., Han, A. W., Johnson, A. J. A., and Holmes, S. P. (2016) DADA2: high-resolution sample inference from Illumina amplicon data. *Nature Methods,* **13**(7), 581.

[23] Clemente, J. C., Pehrsson, E. C., Blaser, M. J., Sandhu, K., Gao, Z., Wang, B., Magris, M., Hidalgo, G., Contreras, M., Noya-Alarcón, Ó., et al. (2015) The microbiome of uncontacted Amerindians. *Science Advances,* **1**(3), e1500183.

[24] Döring, A., Weese, D., Rausch, T., and Reinert, K. (2008) SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics,* **9**(1), 11.

[25] Kingma, D. P. and Ba, J. (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980,*.

[26] Beye, M., Fahsi, N., Raoult, D., and Fournier, P.-E. (2018) Careful use of 16S rRNA gene sequence similarity values for the identification of Mycobacterium species. *New Microbes and New Infections,* **22**, 24–29.