# dna2vec: Consistent vector representations of variable-length k-mers

Patrick Ng
ppn3@cs.cornell.edu

**Abstract**

One of the ubiquitous representation of long DNA sequence is dividing it into shorter k-mer components. Unfortunately, the straightforward vector encoding of k-mer as a one-hot vector is vulnerable to the curse of dimensionality. Worse yet, the distance between any pair of one-hot vectors is equidistant. This is particularly problematic when applying the latest machine learning algorithms to solve problems in biological sequence analysis. In this paper, we propose a novel method to train distributed representations of variable-length k-mers. Our method is based on the popular word embedding model *word2vec*, which is trained on a shallow two-layer neural network. Our experiments provide evidence that the summing of dna2vec vectors is akin to nucleotides concatenation. We also demonstrate that there is correlation between Needleman-Wunsch similarity score and cosine similarity of dna2vec vectors.

## 1 Introduction

The usage of k-mer representation has been a popular approach in analyzing long sequence of DNA fragments. The k-mer representation is simple to understand and compute. Unfortunately, its straightforward vector encoding as a one-hot vector (i.e. bit vector that consists of all zeros except for a single dimension) is vulnerable to curse of dimensionality. Specifically, its one-hot vector has dimension exponential to the length of $k$. For example, an 8-mer needs a bit vector of dimension $4^8 = 65536$. This is problematic when applying the latest machine learning algorithms to solve problems in biological sequence analysis, due to the fact that most of these tools prefer lower-dimensional continuous vectors as input (Suykens and Vandewalle, 1999; Angermueller et al., 2016; Turian et al., 2010). Worse yet, the distance between any arbitrary pair of one-hot vectors is equidistant, even though `ATGGC` should be closer to `ATGGG` than `CACGA`.

### 1.1 Word embeddings

The Natural Language Processing (NLP) research community has a long tradition of using bag-of-words with one-hot vector, where its dimension is equal to the vocabulary size. Recently, there has been an explosion of using *word embeddings* as inputs to machine learning algorithms, especially in the deep learning community (Mikolov et al., 2013b; LeCun et al., 2015; Bengio et al., 2013). Word embeddings are vectors of real numbers that are distributed representations of words.

A popular training technique for word embeddings, word2vec (Mikolov et al., 2013a), consists of using a 2-layer neural network that is trained on the current word and its surrounding context words (see Section 2.3). This reconstruction of context of words is loosely inspired by the linguistic concept of *distributional hypothesis*, which states that words that appear in the same context have similar meaning (Harris, 1954). Deep learning algorithms applied with word embeddings have had dramatic improvements in the areas of machine translation (Sutskever et al., 2014; Bahdanau et al., 2014; Cho et al., 2014), summarization (Chopra et al., 2016), sentiment analysis (Kim, 2014; Dos Santos and Gatti, 2014) and image captioning (Vinyals et al., 2015).

One of the most fascinating properties of word2vec is that its vector arithmetic can solve semantic and linguistic analogies (Mikolov et al., 2013c,a). They showed that $vec(king) - vec(man) + vec(woman) \approx vec(queen)$. In particular, the analogy task `man:king :: woman:???` is interpreted as finding a word $w$ such that $vec(king) - vec(man) + vec(woman)$ is closest to $vec(w)$ under cosine distance. Furthermore, (Levy et al., 2014) showed that analogy works for past-tense relation $vec(capture) - vec(captured) \approx vec(go) - vec(went)$,

language-spoken-in relation $vec(france) - vec(french) \approx vec(mexico) - vec(spanish)$, as well as geographical location $vec(Berlin) - vec(Germany) \approx vec(Paris) - vec(France)$.

### 1.2 dna2vec: k-mer embeddings

In this paper, we present a novel method to compute distributed representations of variable-length k-mers. These k-mers are consistent across different lengths, i.e. they lie in the same embedding vector space. We embed k-mers of length $3 \leq k \leq 8$, which is a space consists of one k-mer per dimension ($d = \sum_{k=3}^{8} 4^k$), into a continuous vector space of 100 dimensions.

The training method of our shallow two-layer neural network for dna2vec is based on word2vec. BioVec (Asgari and Mofrad, 2015) and seq2vec (Kimothi et al., 2016) have also applied the word2vec technique to biological sequences. Although both techniques used a two-layer neural network to train their embedding, our technique is a generalization for variable-length $k$.

(Needleman and Wunsch, 1970) presented a method, now commonly known as Needleman-Wunsch algorithm, for computing similarity of k-mers using a dynamic programming scoring of global alignments. But the dynamic programming nature of the algorithm makes the algorithm slow, with quadratic time complexity to the length of the sequence. In Section 3.3, we show that its cosine distance, in other words angular distance, is related to Needleman-Wunsch distance of their corresponding k-mers. In Section 3.4, we provide evidence that nucleotide concatenation analogy can be constructed with dna2vec arithmetic.

The main contribution of this work includes:

- variable-length k-mer embedding model
- experimental evidence that shows arithmetic of dna2vec vectors is akin to nucleotides concatenation
- relationship between Needleman-Wunsch alignment and cosine similarity of dna2vec vectors
- nucleotide concatenation analogy can be constructed with dna2vec arithmetic.

## 2 Training dna2vec model

The training of dna2vec consists of four stages:

1) separate genome into long non-overlapping DNA fragments
2) convert long DNA fragments into overlapping variable-length k-mers
3) unsupervised training of an aggregate embedding model using a two-layer neural network
4) decompose aggregated model by k-mer lengths.

### 2.1 Stage 1: Long non-overlapping DNA fragments

We fragment the genome sequence based on gap characters (e.g. `X`, `-`, etc). For our experiments using hg38 dataset, the fragments were typically a couple of thousand nucleotides. To introduce more entropy, we randomly choose to use the fragment's reverse-complement.

### 2.2 Stage 2: Overlapping variable-length k-mers

Given a DNA sequence $S$, we convert the sequence $S$ into overlapping fixed length k-mer by sliding a window of length $k$ across $S$. For example, we convert `TAGACTGTC` into five 5-mers: `{TAGAC, AGACT, GACTG, ACTGT, CTGTC}`. In the variable-length case, we sample $k$ from the discrete uniform distribution $Uniform(k_{low}, k_{high})$ to determine the size of each window. For example, a sample of k-mers of $k \in \{3, 4, 5\}$ could be `{TAGA, AGA, GACT, ACT, CTGTC}`.

Formally, given a sequence of length $n$, $S = (S_1, S_2, ..., S_n)$ where $S_i \in \{A, C, G, T\}$, we convert $S$ into $\tilde{n} = n - k_{high} + 1$ number of k-mers:

$$f(S) = (S_{1:k_1}, S_{2:2+k_2}, ...S_{\tilde{n}:\tilde{n}+k_{\tilde{n}}})$$
$$k_i \sim Uniform(k_{low}, k_{high})$$

where $S_{a:b}$ is a shorthand for $(S_a, ..., S_b)$.

### 2.3 Stage 3: Two-layer neural network

We use a shallow two-layer neural to train an aggregate DNA k-mer embedding. The method is based on word2vec (Mikolov et al., 2013a). The word2vec algorithm has the options of continuous bag-of-words (CBOW) or skip-gram. CBOW predicts the targeted word given the context, while skip-gram predicts the context given the targeted word. The word2vec homepage[1] claims that skip-gram is slower to train than CBOW, but skip-gram is better for infrequent words. We use skip-gram for all our experiments.

Our dna2vec algorithm is trained by predicting the "context" surrounding a given targeted k-mer. The "context" is the set of adjacent k-mers surrounding the targeted k-mer. For example, the context of k-mer `GACT` would be `{TAGA, AGA, ACT, CTGTC}` in our previous example from Section 2.2. For our experiments in this paper, we used a context size of 10 before and after the targeted word, which amounts to predicting a total of 20 k-mers.

During training, either negative sampling or hierarchy softmax is typically used to optimize the update procedure over all words. We used negative sampling for all our experiments.

### 2.4 Stage 4: Decompose aggregated model by k-mer lengths

We decompose the aggregate model by k-mer length to form $k_{high} - k_{low} + 1$ models. This decomposition is useful for the searching of nearest neighbors, as we will discuss in Section 3.1.

## 3 Experiments

Our dna2vec[2] was trained with hg38 human assembly chr1 to chr22 (Rosenbloom et al., 2015). Specifically, they were downloaded from http://hgdownload.cse.ucsc.edu/downloads.html#human. We excluded X and Y chromosomes, as well as mitochondrial and unlocalized sequences.

### 3.1 Similarity and nearest neighbors

For each vector arithmetic solution, we often compute its $n$-nearest k-mer neighbors. We define *similarity* between two dna2vec vectors $v, w \in \mathbb{R}^d$ as the cosine similarity:

$$sim(v, w) = \frac{v \cdot w}{\|v\|\|w\|}$$

The *nearest-neighbor* of dna2vec vector $v \in \mathbb{R}^d$ is a k-mer computed with:

$$NearestNeighbor_k(v) = \underset{s \in \{A,C,G,T\}^k}{\arg\max} \; sim(v, vec(s)) \tag{1}$$

[1] https://code.google.com/archive/p/word2vec/

[2] pre-trained dna2vec vectors available at https://pnpnpn.github.io/dna2vec/ upon publication

Generally, the $nNearestNeighbors_k(v)$ are the $n$-nearest neighboring k-mers to vector $v$.

### 3.2 dna2vec arithmetic and nucleotide concatenation

We found that summing dna2vec embeddings is related to concatenating k-mers. In Table 1, we investigated this hypothesis by adding dna2vec embeddings of two arbitrary k-mers and examining whether their vector sum's neighbors overlap with their string concatenation. The *1-NN* column results were tallied using Equation (1) and the other columns used *nNearestNeighbors* from Section 3.1. In this experiment, string concatenation can come from both 5' and 3' ends. For example, the following condition would be marked as a *success* for *1-NN*:

$$NearestNeighbor_6(vec(\texttt{AAC}) + vec(\texttt{TCT})) \in \{\texttt{AACTCT}, \texttt{TCTAAC}\}$$

Likewise, the following would be a *success* for *n-NN*:

$$nNearestNeighbors_6(vec(\texttt{AAC}) + vec(\texttt{TCT})) \cap \{\texttt{AACTCT}, \texttt{TCTAAC}\} \neq \emptyset$$

Table 1: K-mers concatenation and dna2vec addition. We took 1000 samples for each operand. For example, the first row is aggregated from summing the dna2vec vectors of individual pairs of arbitrary 3-mer and observing whether each of their string concatenation overlaps with the vector sum's $n$-nearest 6-mer neighbors.

| **Operands** | **Concatenated** | **1-NN** | **5-NN** | **10-NN** |
|---|---|---|---|---|
| 3-mer + 3-mer | 6-mer | 28.7% | 80.3% | 94.6% |
| 3-mer + 4-mer | 7-mer | 49.9% | 90.4% | 97.4% |
| 3-mer + 5-mer | 8-mer | 53.9% | 94.0% | 98.4% |
| 4-mer + 4-mer | 8-mer | 73.5% | 96.8% | 99.2% |

### 3.3 Relationship to global alignment similarity

All of the Needleman-Wunsch similarity score in this paper were computed using Biopython's `align.globalxx` function, which used a match score of 1, mismatch of 0 and gap penalty of 0.

In Figure 1, we provided evidence that edit distance between two arbitrary k-mers is correlated with the cosine distance of their corresponding dna2vec vectors. We sampled 1000 pairs of 8-mers for each Needleman-Wunsch score level and plot their Needleman-Wunsch similarity score against dna2vec cosine similarity.

In Figure 2, we compared the Needleman-Wunsch similarity distribution of k-mer and its nearest dna2vec neighbor against distribution of two random k-mers. Specifically, we sampled 1000 8-mers, found each of its nearest neighbor using Equation (1), and computed the Needleman-Wunsch score for each pair. For the null distribution, we sampled 1000 pairs of random 8-mers. Thus we found evidence that the dna2vec nearest-neighbor exhibits alignment similarity.

### 3.4 Analogy of nucleotide concatenation

We experimented with two types of nucleotide concatenation analogy: *strong* and *weak* concatenations. Given two k-mers of the same length, we define *strong concatenation* as splicing nucleotides on the same end (either 5' or 3' end) of the k-mers. An example of 5-mer with 3-nucleotides snippet would be:

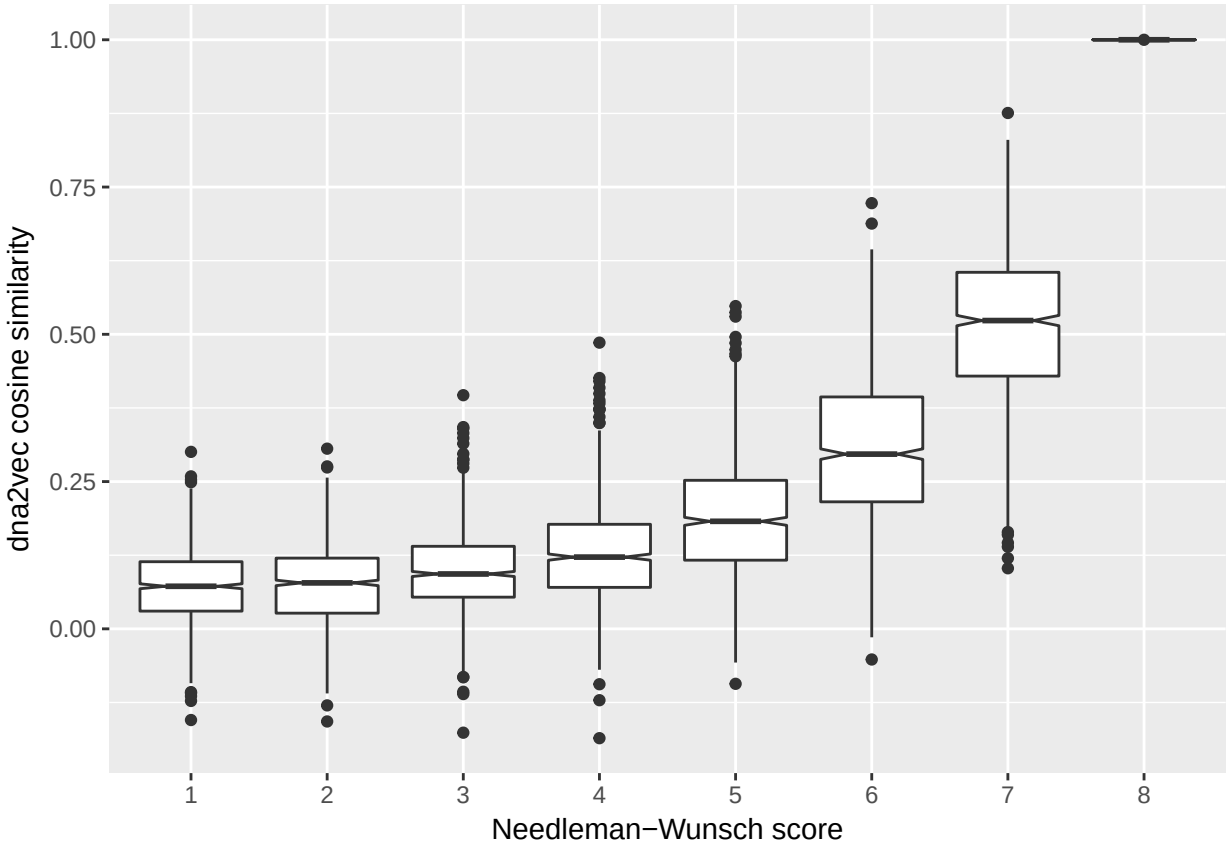


Figure 1: Boxplot of Needleman-Wunsch score and dna2vec cosine similarity. The lower and upper hinges are the 25 and 75 quartiles, respectively. The Spearman's rank correlation coefficient is 0.831

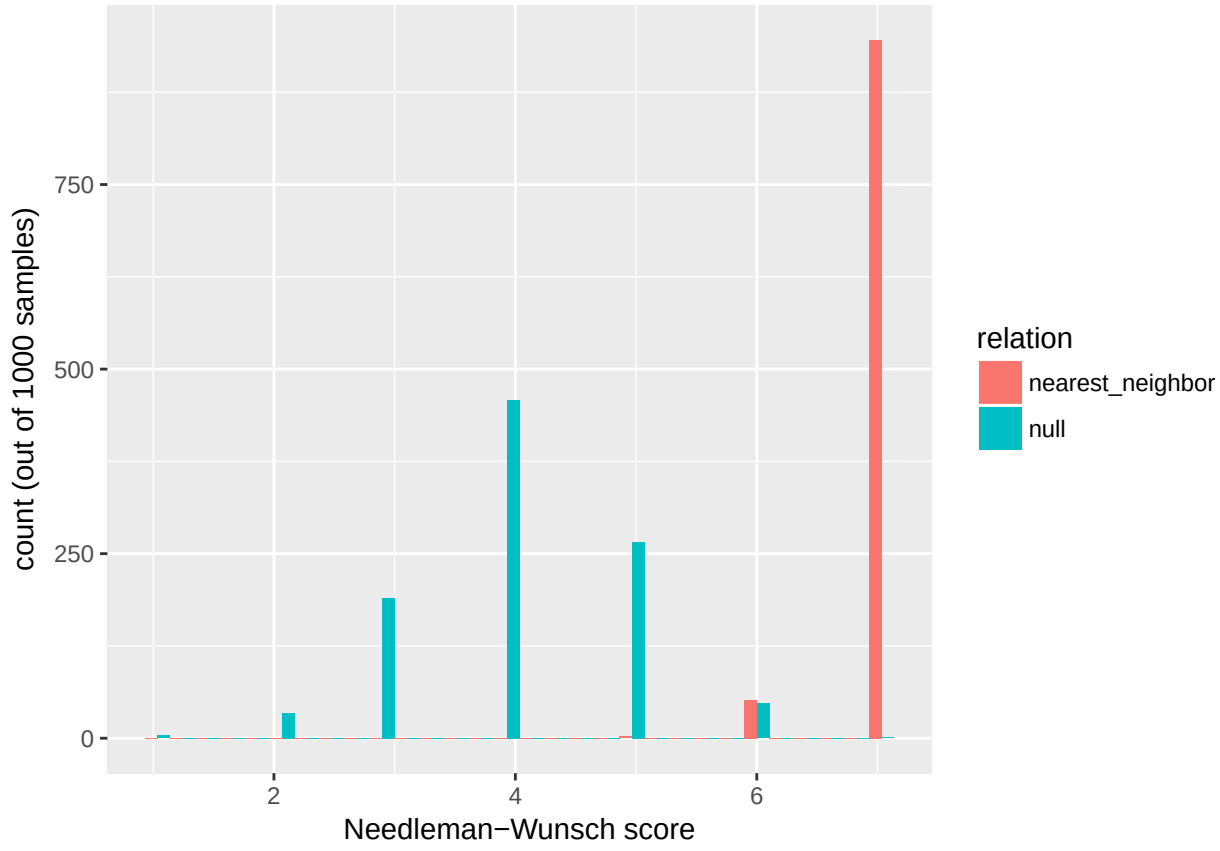


Figure 2: Global alignment score distribution of nearest-neighbor. The `nearest-neighbor` distribution is generated by computing Needleman-Wunsch score between 8-mer and its nearest neighbor. The `null` distribution is from computing the score between two random 8-mers.

$$vec(\mathbf{AC}GAT) - vec(GAT) + vec(ATC) \approx vec(\mathbf{AC}ATC)$$

We relaxed the *same end* restriction in the experiment *weak concatenation*, i.e. the result can come from either end:

$$vec(\mathbf{AC}GAT) - vec(GAT) + vec(ATC) \in_{approx} \{vec(\mathbf{AC}ATC), vec(ATC\mathbf{AC})\}$$

Experimental samples were generated from randomly sampling two k-mers of equal length and a nucleotide snippet (3 or 4 nucleotides) for concatenation. For both strong and weak concatenation experiments, we randomly selected either the 5' and 3' end to splice.

Table 2 shows the summary of experimental results from the two types of nucleotide concatenations. Particularly, we get 88% accuracy with weak concatenation analogy of 8-mer and 4-nucleotides snippet when considering 10-NN, as defined in Section 3.1. Note that considering 30-nearest neighbors is relatively small comparing to the space of all possible 6-mers, it is merely 0.73% of all possible 6-mers and 0.046% of all possible 8-mers.

To confirm whether the arithmetic was actually extending a k-mer by the snippet as oppose to similarity comparison, we compared the analogy results with *scrambled-snippet* experiments, which concatenated a different random snippets in the answer case. As expected, the vector arithmetic was significantly favoring the correct matching snippet (`analogy` column) over a different random snippet (`scrambled-snippet` column) in Figure 3 and Table 2.

Table 2: Analogy Experiment. We analyzed two types of analogies: *weak* and *strong* concatenation. 1000 samples were randomly generated for each type. For comparison, we generated 1000 samples using scrambled-snippet sampling strategy.

| dimension | weak-concat scrambled-snippet 5 / 10 / 30-NN | weak-concat analogy 5 / 10 / 30-NN | strong-concat scrambled-snippet 5 / 10 / 30-NN | strong-concat analogy 5 / 10 / 30-NN |
|---|---|---|---|---|
| 6-mer with 3-nt snippet | *1.4 / 4 / 16%* | 47 / 69 / 95% | *0.6 / 1.8 / 9%* | 43 / 62 / 88% |
| 7-mer with 3-nt snippet | *2.4 / 6 / 16%* | 66 / 82 / 96% | *1.5 / 3.8 / 10%* | 61 / 76 / 92% |
| 8-mer with 3-nt snippet | *3 / 6 / 19%* | 67 / 82 / 95% | *2.3 / 3.8 / 11%* | 62 / 77 / 91% |
| 8-mer with 4-nt snippet | *0.7 / 1.4 / 3%* | 75 / 88 / 98% | *0.3 / 1.0 / 2.4%* | 69 / 83 / 95% |

### 3.5 Implementation Details

We will make our code and data available at https://pnpnpn.github.io/dna2vec/ upon publication. The two-layer neural network training method described in Section 2.3 was implemented using `gensim` framework (Řehůřek and Sojka, 2010). We used gensim's `Word2vec` class with parameters `sg=1` and `window=10`, which specified the usage of skipgram model and the half-size of the context window as 10, respectively. All of trained dna2vec vectors used in this paper has dimension size of 100. Since the window sliding step in Section 2.2 is stochastic in terms of variable-length $k$, we could essentially generate more training data by looping through the complete genomic sequence data with multiple passes, which we called *epochs*. The dna2vec model used in this paper was trained with 10 epochs. The training step took over 3 days using gensim parameter `workers=4` on a 2.66 GHz Quad-Core Intel Xeon with 8GB memory.

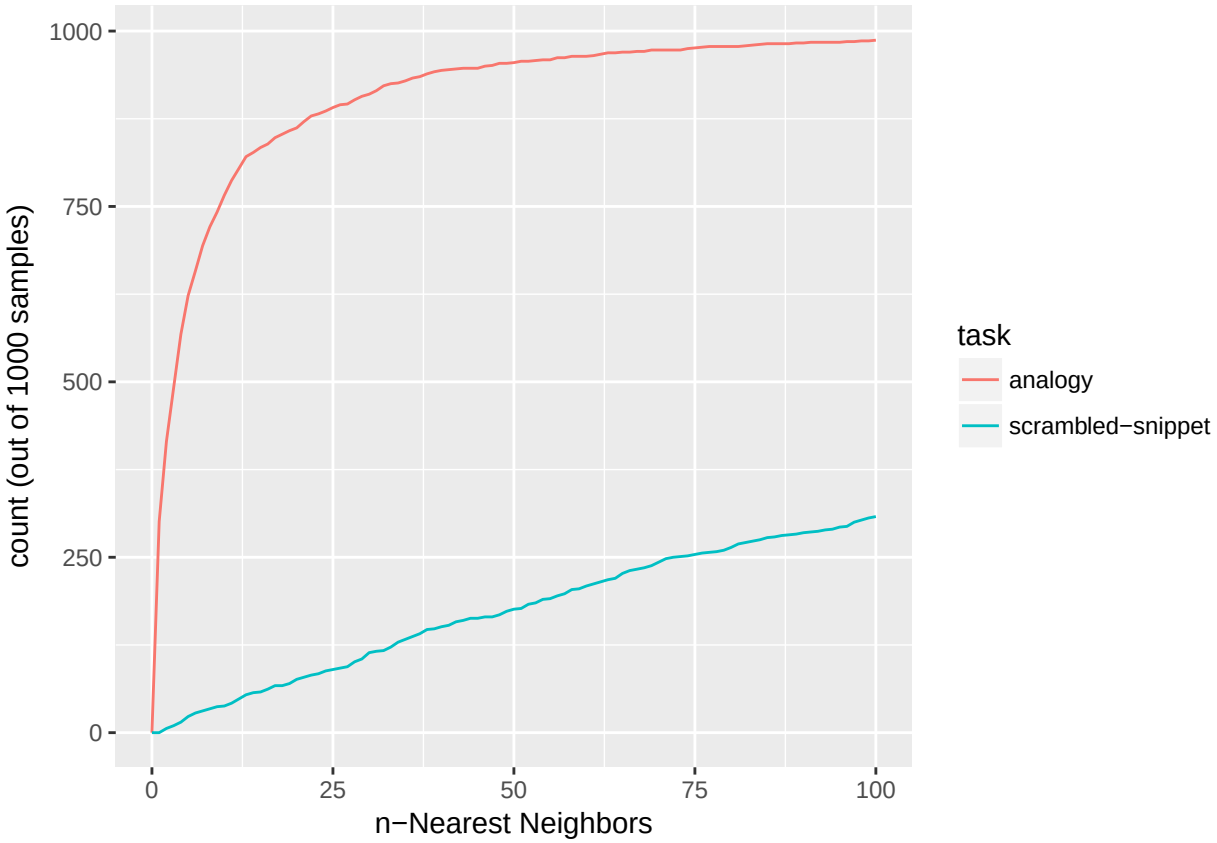


Figure 3: Cumulative mass for analogy experiment of 8-mer with 3-nt snippet. 1000 samples were generated with the strong-concatenation analogy setup. We compared it with another 1000 samples using the *scrambled-snippet* sampling procedure.

# 4 Discussion

In this work, we presented a novel method for training distributed representations of k-mers. We demonstrated that our dna2vec embeddings can represent variable-length k-mers in a consistent fashion via nucleotide concatenation experiments. We provided experimental evidence showing that the arithmetic of dna2vec vectors is akin to nucleotides concatenation. We also showed that Needleman-Wunsch similarity score between two arbitrary k-mers is correlated with the cosine distance of their corresponding dna2vec vectors. As for future work, due to the fact that many machine learning algorithms require fixed-length continuous vectors as input, we will explore the application of dna2vec with machine learning techniques on biological sequence analysis.

# References


Angermueller, C., Pärnamaa, T., Parts, L., and Stegle, O. (2016). Deep learning for computational biology. *Molecular systems biology*, 12(7):878.

Asgari, E. and Mofrad, M. R. (2015). Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11):e0141287.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chopra, S., Auli, M., Rush, A. M., and Harvard, S. (2016). Abstractive sentence summarization with attentive recurrent neural networks. *Proceedings of NAACL-HLT16*, pages 93–98.

Dos Santos, C. N. and Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kimothi, D., Soni, A., Biyani, P., and Hogan, J. M. (2016). Distributed representations for biological sequence analysis. *arXiv preprint arXiv:1608.05949*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Levy, O., Goldberg, Y., and Ramat-Gan, I. (2014). Linguistic regularities in sparse and explicit word representations. In *CoNLL*, pages 171–180.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *HLT-NAACL*, volume 13, pages 746–751.

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.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Rosenbloom, K. R., Armstrong, J., Barber, G. P., Casper, J., Clawson, H., Diekhans, M., Dreszer, T. R., Fujita, P. A., Guruvadoo, L., Haeussler, M., et al. (2015). The ucsc genome browser database: 2015 update. *Nucleic acids research*, 43(D1):D670–D681.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Suykens, J. A. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300.

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164.