Dual Descriptor for Sequence Analysis: A Revisit

Bin-Guang Ma[*]

Hubei Key Laboratory of Agricultural Bioinformatics, College of Informatics, Huazhong Agricultural University, Wuhan 430070, China

[*] Corresponding author. Tel & Fax: +86 2787280877.
E-mail address: mbg@mail.hzau.edu.cn (Bin-Guang Ma)

Abstract

The Dual Descriptor (DD) method, originally introduced for biological sequence analysis, is revisited and refined in this article. DD captures both compositional and permutational information of sequences through a Composition Weight Map (CWM) and a Position Weight Function (PWF). This article elaborates on its formulation, including scalar, vector, numeric, and hierarchical forms, and presents multiple implementation variants. Training involves minimizing a pattern deviation function, with closed-form or gradient-based optimization. Applications to sequence problems span feature extraction, identification, classification, regression, and sequence generation. An object-oriented Python implementation is provided for the demonstration purpose of methodology. DD offers a flexible, mathematically grounded framework for sequence modeling, with potential integration into broader machine learning pipelines.

Keywords: Dual Descriptor (DD); Numeric DD; Hierarchical DD; Linker matrix; DD network

1. Introduction

Dual Descriptor (DD) method was originally proposed in the master's thesis of Bin-Guang Ma (Ma, 2003) and partially described in two articles (Ma, 2007; Ma, 2008). In this article, we revisit and refine its formulation and apply this methodology to more sequence processing problems. A Python implementation of this methodology for demonstration purpose is available at: https://github.com/mbglab/DualDescriptor, which is referred to as the **base** directory in the following.

2. Formulation

2.1. Definition of Dual Descriptor

As elaborated in Ma (2008), for a character set $C=\{c_1,c_2,\cdots,c_i,\cdots,c_n\}$ $(n\geq 2)$, we use *C** to represent the set of the sequences composed of the characters in *C* and with finite lengths. On *C**, Dual Descriptor (DD) is defined as a two-element set:

$$DD=\{M,\ P\} \tag{1}$$

where *M* is the Composition Weight Map (CWM) and *P* is the Position Weight Function (PWF) which are used jointly to reflect the two aspects of information of a character sequence: composition and permutation.

M is a map from the character set *C* to a real number set *X*, i.e., $M:C\to X$, $X=\{x_1,x_2,\cdots,x_i,\cdots,x_n\,|\,(x_i\in R—\{0\})\}$. *P* is a real-valued function of position *k* in the character sequence *s*; *P*(*k*) reflects the endued weight to the position *k*.

2.1.1. Pattern Description Function

For a sequence *s* composed of the characters in *C* with length *L*, under the map $M:C\to X$, it can be converted into a real number sequence *x*, namely:

$$\begin{gathered} s=[s[1],s[2],\cdots,s[k],\cdots,s[L]] \\ \Downarrow \\ x=[x[1],x[2],\cdots,x[k],\cdots,x[L]] \end{gathered} \tag{2}$$

where $x[k]=x_i$ if $s[k]=c_i$ $(k=1,2,\cdots,L;\ x_i\in X,\ c_i\in C)$.

For the character sequence *s*, its pattern description function is defined as:

$$N(k)=P(k)\times x[k] \quad (k=1,2,\cdots,L) \tag{3}$$

where the coefficient $P(k)$ before $x[k]$ is the position weight function.

2.1.2. Dual Formula

The sum of the first *l* items of *N*(*k*) is

$$S(l)=\sum_{k=1}^{l}N(k)=\sum_{k=1}^{l}P(k)x[k]=\sum_{x_i\in X}x_i\sum_{k_{x_i}}P(k_{x_i}) \tag{4}$$

where k_{x_i} represents the position *k* where x_i appears. *S*(*l*) indicates some kind of dual relation and thus is called Dual Formula or Dual Variable. Let $P_{x_i}=\sum_{k_{x_i}}P(k_{x_i})$; P_{x_i} $(x_i\in X)$ is the position-weighted frequencies when it is normalized by the sequence length *L*, which constitutes the "permutation part" of dual variable. x_i $(x_i\in X)$ are called Composition Weight Factors (CWF), which constitute the "composition part" of dual variable. These two parts are interdependent on each other and jointly reflect the information of a character sequence.

2.1.3. Target Pattern and Standard Pattern

When $P(k)=\text{constant}$ and $x[k]=\text{constant}$, the pattern description function is also a constant: $N(k)=P(k)x[k]=\text{constant}=t$ $(k=1,2,\cdots,L)$, and *t* is called Target Pattern. When *t* = 1, namely, $N(k)=1$ $(k=1,2,\cdots,L)$, it is called Standard Pattern.

2.2. Training DD to describe patterns of character sequence

Dual Descriptor can be trained on datasets. The training process of DD is the process of feature extraction from character sequence, which is implemented by minimizing the pattern deviation of a character sequence from a target pattern.

2.2.1. Pattern Deviation Function

To describe the pattern deviation of a sequence, we defined the Pattern Deviation Function (PDF) as:

$$d = \frac{1}{L}\sum_{k=1}^{L}\left(N(k)-t\right)^2 \tag{5}$$

which represents the deviation of a sequence (whose pattern is described by *N*(*k*)) from a target pattern *t*. when $t=1$, *d* represents the deviation of the sequence from the standard pattern: $N(k)=1 \quad (k=1,2,\cdots,L)$.

2.2.2. Minimization of Pattern Deviation Function

The training of a DD is to minimize *d*. Substitute Eq. (3) into Eq. (5), we get

$$d = \frac{1}{L}\sum_{k=1}^{L}\left(P(k)x[k]-t\right)^2 . \tag{6}$$

P(*k*) can be expanded on a set of basis functions $b_h(k) \ \ (h=1,\ 2,\ \cdots,\ o)$, i.e.,

$$P(k) = \sum_{h} a_h b_h(k) \quad (h=1,\ 2,\ \cdots,\ o), \tag{7}$$

in which a_h is independent of *k* and $b_h(k) \ \ (h=1,\ 2,\ \cdots,\ o)$ is the *o* items of the basis functions. The coefficients (a_h) in the expanded form of the position weight function *P*(*k*) are abbreviated as PWC (Position Weight Coefficients).

One-step training from C: For a given CWM, $x_i \quad (x_i \in X_0)$ are constants. To minimize *d*, from $\frac{\partial d}{\partial a_h} = 0$, we get

$$\begin{aligned} u_{hg} &= \sum_{k=1}^{L} b_h(k) b_g(k) x[k]^2 \\ v_h &= t\sum_{k=1}^{L} b_h(k) x[k] \end{aligned} \qquad \left(h,g=1,\ 2,\ \cdots,\ o\right) \tag{8}$$

where $b_h(k)$ and $b_g(k)$ are the *h*-th and *g*-th basis functions, respectively, and $x[k] \in X_0$ is the number at the *k*-th position in the real number sequence *x*. The coefficients of *P*(*k*) can be written as a vector **a** which can be obtained from the matrix **u** and the vector **v**:

$$\mathbf{a} = \mathbf{u}^{-1}\mathbf{v} \tag{9}$$

where $\mathbf{a} = (a_1,\ a_2,\ \cdots,\ a_h,\ \cdots,\ a_o)$ and the matrix **u** and the vector **v** are composed of the elements u_{hg} and v_h.

One-step training from P: For a given PWF ($P_0(k)$), $a_h(k)$ $(h=1,\ 2,\ \cdots,\ o)$ are constants. To minimize *d*, from $\frac{\partial d}{\partial x_i}=0$, we arrive at:

$$x_i=\frac{t\sum_{k_{x_i}}P_0(k_{x_i})}{\sum_{k_{x_i}}P_0^2(k_{x_i})} \tag{10}$$

where k_{x_i} represents the position *k* in the real number sequence *x* where x_i appears.

2.2.3. Extracting common features of multiple sequences

For the situation of multiple sequences, to extract the common features of these sequences, the PDF for the ensemble of these sequences is defined as the mean value of the PDF values of these sequences:

$$D=\frac{1}{N}\sum_{j=1}^{N}d_j \tag{11}$$

where *N* is the number of the sequences, and $d_j=\frac{1}{L_j}\sum_{k=1}^{L_j}(N_j(k)-t_j)^2$ is the PDF value for the *j*-th sequence with the length *L*L_j, and $N_j(k)$ and t_j are the pattern description function and target of the *j*-th sequence.

One-step training from C: For a given CWM, from $\frac{\partial D}{\partial a_h}=\sum_{j=1}^{N}\frac{\partial d_j}{\partial a_h}=0$, we arrive at:

$$\begin{matrix} U_{hg}=\sum_{j=1}^{N}u_{hg}^{j} \\ V_h=\sum_{j=1}^{N}v_h^{j} \end{matrix} \quad \left(h,g=1,\ 2,\ \cdots,\ o\right). \tag{12}$$

At this time, the coefficient vector **a** is given by

$$\mathbf{a}=\mathbf{U}^{-1}\mathbf{V} \tag{13}$$

where the matrix **U** and the vector **V** are the sums of **u** and **v** (that are for single sequence), respectively.

One-step training from P: Similarly, for a given PWF, from $\frac{\partial D}{\partial x_i}=\sum_{j=1}^{N}\frac{\partial d_j}{\partial x_i}=0$, we arrive at:

$$x_i = \frac{t_j \sum_j \sum_{k_{x_i}} P_0(k_{x_i})}{\sum_j \sum_{k_{x_i}} P_0^2(k_{x_i})} . \tag{14}$$

2.2.4. Alternate training process

The alternate training process for a DD on a sequence (or a set of sequences) consists of the following steps:

Step (1): Preparing a dataset composed of one or multiple sequences; set the maximum step number $T_{\max}$.

Step (2): Randomly construct a CWM, i.e, assign a random real number to each of the characters in C and these real numbers constitute the set of $x_i \quad (x_i \in X)$, and then use the minimization condition in Eq. (9) or Eq. (13) to obtain a corresponding PWF, namely, a set of coefficients $a_h(k) \ \ (h=1,\ 2,\ \cdots,\ o)$.

Step (3): With the set of coefficients $a_h(k) \ \ (h=1,\ 2,\ \cdots,\ o)$, use the minimization condition in Eq. (10) or Eq. (14) to obtain a CWM, namely, a set of $x_i \quad (x_i \in X)$; Generally speaking, the set of $x_i \quad (x_i \in X)$ obtained in this step is not the same as those used in Step (1).

Step (4): Repeat Step (2) and Step (3) until the stop condition is satisfied.

Stop condition: the minimum *d* (or *D*) is achieved, or the maximum step number $T_{\max}$ is reached.

In the training process, *d* (or *D*) becomes smaller and smaller. When *d* (or *D*) reaches its minimum value, the training process stops and an optimum DD is obtained on the dataset of the sequences used. Dual descriptor method can be viewed as a kind of machine-learning approach. Different from other machine-learning approaches where local minimums are ubiquitous and cannot be tackled readily, the DD method does not yield local minimum in principle because the PDF (Eq. (5)) is a quadric function and has only a unique global minimum.

2.2.5. Gradient-based training

In addition to the alternate training process where DD is trained by solving closed-form equation systems, DD can also be trained by commonly used gradient descent method, particularly for large systems (big *n* and *o*) or the vector form of DD (see below). The gradients are computed as:

$$\begin{aligned}\frac{\partial D}{\partial a_h} &= \frac{2}{N}\sum_{j=1}^{N}\sum_{k=1}^{L}\left(N_j(k)-t_j\right)\cdot b_h(k)\cdot x_i^j[k] \\ \frac{\partial D}{\partial x_i} &= \frac{2}{N}\sum_{j=1}^{N}\sum_{k=1}^{L}\left(N_j(k)-t_j\right)\cdot P(k)\end{aligned}, \tag{20}$$

and parameters are updated according to

$$\begin{aligned}a_h' &= a_h - \eta\cdot\frac{\partial D}{\partial a_h} \\ x_i' &= x_i - \eta\cdot\frac{\partial D}{\partial x_i}\end{aligned}, \tag{21}$$

where η is the learning rate.

2.2.6. Predicting target

After training, DD carries the information (building patterns) of the character sequences in the training dataset. For a new character sequence, trained DD can be used to predict its expected target value. The formula for target prediction is:

$$t_{pred} = \frac{1}{L}\sum_{k=1}^{L} N(k) = \frac{1}{L}\sum_{k=1}^{L} P(k)\cdot x[k], \tag{22}$$

which is the dual variable value normalized by sequence length calculated based on the trained DD and the new sequence.

2.2.7. Choice of the basis functions

Basis functions in Eq. (7) are usually chosen as periodic functions, such as trigonometric function

$$P(k) = \sum_h a_h \cos\left(\frac{2\pi k}{h}\right) \qquad (h = 1 \text{ or } 2,\ 3,\ \cdots,\ o) \tag{23}$$

or

$$P(k) = e^{k \bmod h} \qquad (h = 1 \text{ or } 2,\ 3,\ \cdots,\ o) \tag{24}$$

because periodicity is a main difference between an ordered and a random sequence. Note that *h* starts from

1 (usually for one-step training) or 2 (usually for alternate training). Except for trigonometric functions, basis functions can also be other types of functions like wavelet functions, radial basis function, sigmoid functions, *etc.*, depending on question contexts. A Python implementation of scalar DD class is presented at: **base**/DD/DDs.py.

2.3. The vector form of DD

2.3.1. Encoding vectors and position weight function matrix

As described in Ma (2003), The CWM $M:C \to X$ can be generalized into $M:C \to \mathbf{X}$, where $\mathbf{X} = \left\{ \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_i, \cdots, \mathbf{x}_n \mid \mathbf{x}_i \in R^m \right\}$ is a set of *m*-dimensional real vectors. The vectors in the vector set **X** are column vectors, referred to as "encoding vectors". Arranging these *n* column vectors of dimension *m* from the vector set **X** together forms a matrix $M_{m \times n}$, referred to as the "encoding matrix", i.e.,

$$M_{m\times n} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1i} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2i} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{j1} & x_{j2} & \cdots & x_{ji} & \cdots & x_{jn} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mi} & \cdots & x_{mn} \end{bmatrix} \quad (x_{ji} \in R - \{0\}; i = 1,2,\cdots,n, j = 1,2,\cdots,m) . \tag{25}$$

Each column in this matrix corresponds to one character in the character set *C*, and each row corresponds to one component of the vectors. To reduce information redundancy, the components should be mutually independent, meaning the row vectors of the "encoding matrix" should be linearly independent from each other; ideally, they should be mutually orthogonal. However, this principle is not an absolute requirement.

Under the map $M:C \to \mathbf{X}$, a character sequence of length *L* is transformed into a sequence of *m*-dimensional real vectors:

$$\begin{gathered} s = [s[1], s[2], \cdots, s[k], \cdots, s[L]] \ (s[k] \in C, \ k = 1,2,\cdots,L) \\ \Downarrow \\ \mathbf{x} = [\mathbf{x}[1], \mathbf{x}[2], \cdots, \mathbf{x}[k], \cdots, \mathbf{x}[L]], \quad (\mathbf{x}[k] \in \mathbf{X}) \end{gathered} . \tag{26}$$

The position-weighted sum its first *l* terms, i.e., the dual formula, is as follows:

$$\mathbf{s}(l)=\sum_{k=1}^{l}\mathbf{P}(k)\mathbf{x}[k] \quad (l=1,2,\cdots,L)\,, \tag{27}$$

in which

$$\mathbf{P}(k)=\begin{bmatrix} P_{11}(k) & P_{12}(k) & \cdots & P_{1q}(k) & \cdots & P_{1m}(k) \\ P_{21}(k) & P_{22}(k) & \cdots & P_{2q}(k) & \cdots & P_{2m}(k) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ P_{p1}(k) & P_{p2}(k) & \cdots & P_{pq}(k) & \cdots & P_{pm}(k) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ P_{m1}(k) & P_{m2}(k) & \cdots & P_{mq}(k) & \cdots & P_{mm}(k) \end{bmatrix}_{m\times m} \tag{28}$$

is the Position Weight Function Matrix (PWFM, an *m* square matrix), where each element $P_{pq}(k)$ is a positional weight function. **x**[*k*] is the *k*-th vector (of dimension *m*) in the vector sequence **x**, corresponding to the *k*-th character in the character sequence *s*:

$$\mathbf{x}[k]=\mathbf{x}_i \;\; \text{if} \;\; (s[k]=c_i) \;\; (k=1,\ 2,\ \ldots,\ L;\ \mathbf{x}_i\in\mathbf{X},\ c_i\in C)\,. \tag{29}$$

From Eq. (27), when *l* varies from 1 to *L*, a sequence **s** of dual variables **s**(*l*) is obtained. This is also a vector sequence, namely, $\mathbf{s}=[\mathbf{s}[1],\mathbf{s}[2],\cdots,\mathbf{s}[l],\cdots,\mathbf{s}[L]]$. This sequence is the position-weighted sum sequence of the first *l* terms of the encoding vector sequence **x**.

2.3.2. The vector form of PDF

In the vector form of DD, the PDF is defined as:

$$d=\frac{1}{L}\sum_{k=1}^{L}\left\|\mathbf{N}(k)-\mathbf{t}\right\|^2=\frac{1}{L}\sum_{k=1}^{L}\left\|\mathbf{P}(k)\mathbf{x}[k]-\mathbf{t}\right\|^2 \tag{30}$$

where $\mathbf{P}_{m\times m}(k)$ is the position weight function matrix Eq. (28) mentioned earlier, in which each element $P_{pq}(k)$ is a position weight function, and can be expanded into a sum of *o* terms using the basis *b*(*k*) as follows:

$$P_{pq}(k)=\sum_{h=1}^{o}a_h^{pq}b_h^{pq}(k)\,, \tag{31}$$

and **x**[k] is the *k*-th vector (of dimension *m*) in the vector sequence **x**; **t** is the target vector, an *m*-dimensional constant vector.

2.3.3. Alternate Training of vector DD

If the position weight function matrix $\mathbf{P}_{m\times m}(k)$ is known, from $\dfrac{\partial d}{\partial \mathbf{x}_i}=0$ we obtain

$$\mathbf{x}_i=\mathbf{w}^{-1}\mathbf{y} \tag{32}$$

where the elements of matrix **w** and vector **y** are

$$\begin{aligned} w_{pq}&=\sum_{k_{\mathbf{x}_i}}\sum_{f=1}^{m}P_{fp}(k_{\mathbf{x}_i})P_{fq}(k_{\mathbf{x}_i}) \\ y_p&=\sum_{k_{\mathbf{x}_i}}\sum_{f=1}^{m}t_f P_{fp}(k_{\mathbf{x}_i}) \end{aligned} \qquad (f,p,q=1,2,\cdots,m)\,. \tag{33}$$

If $M:C\rightarrow \mathbf{X}$ is known, from $\dfrac{\partial d}{\partial \mathbf{a}_f}=0$ we obtain

$$\mathbf{a}_f=\mathbf{u}_f^{-1}\mathbf{v}_f \tag{34}$$

where $\mathbf{a}_f$ represents the total coefficient row vector for the expansion of all elements in the *f*-th row of the position weight function matrix. This row vector contains $m\times o$ components. These components are grouped every o elements, belonging respectively to the m elements of the *f*-th row. The elements of matrix $\mathbf{u}_f$ and vector $\mathbf{v}_f$ are given by:

$$\begin{aligned} u_{pq}^f&=\sum_{k=1}^{L}b_{hg}^f(k)b_{h'g'}^f(k)x[k]_s x[k]_t \\ v_p^f&=t_f\sum_{k=1}^{L}b_{hg}^f x[k]_s \end{aligned} \qquad (f,p,q=1,2,\cdots,m) \tag{35}$$

where

$$\begin{aligned} h&=\left\lceil \frac{p}{o}\right\rceil \qquad g=\delta(p\bmod o)\cdot o+p\bmod o \\ h'&=\left\lceil \frac{q}{o}\right\rceil \qquad g'=\delta(q\bmod o)\cdot o+q\bmod o\ . \\ s&=\left\lceil \frac{p}{o}\right\rceil \qquad t=\left\lceil \frac{q}{o}\right\rceil \end{aligned} \tag{36}$$

Here, $\lceil\bullet\rceil$ denotes taking the smallest integer not less than "$\bullet$", and $\delta(\bullet)=\begin{cases}1 & \text{if } \bullet=0\\ 0 & \text{if } \bullet\neq 0\end{cases}$.

In the case of multiple sequences, let the number of sequences be *N*, and $D = \frac{1}{N}\sum_{j=1}^{N} d_j$, then we have

$$\begin{aligned} &\mathbf{W} = \sum_{j=1}^{N} \mathbf{w}_j \quad \mathbf{Y} = \sum_{j=1}^{N} \mathbf{y}_j \\ &\mathbf{U}^f = \sum_{j=1}^{N} \mathbf{u}_j^f \quad \mathbf{V}^f = \sum_{j=1}^{N} \mathbf{v}_j^f \end{aligned} . \tag{37}$$

All the matrices above are symmetric matrices. Formally, if all components are mutually independent, the position weight function matrix simplifies to a diagonal matrix of $m \times m$.

2.3.4. Implementations of vector DD

There can be several different implementation forms for vector DD depending on how to deal with PWFM. These implementation forms are special cases of the vector DD. The Python implementation of vector DD classes are presented at: **base**/DD/DDv/.

Form 1: Tensor form: In this form, PWFM is implemented as a 3D tensor $\mathbf{P} \in R^{m \times m \times o}$ and each element in the tensor is a coefficient of a basis function *b*(*k*). **P** is randomly initialized. There are totally $n \times m + m \times m \times o = m(n + m \times o)$ trainable parameters in this form of vector DD, where *n* is the size of character set *C*, *m* is the dimension of encoding vector, and *o* is the number of bases in each expanded position weight function (totally $m \times m$ PWFs). See also: **base**/DD/DDv/ DDvTS.py.

M map: $M : C \to \mathbf{X}$, a learnable matrix $\mathbf{M} \in R^{m \times n}$, randomly initialized

P tensor: a learnable 3D tensor $\mathbf{P} \in R^{m \times m \times o}$ for the coefficients of basis

Basis: $$\begin{aligned} &b_{p,q,h}(k) = \cos\left(\frac{2\pi k}{\text{period}[p,q,h]}\right) \qquad (0 \le p,q < m, \ \ 0 \le h < o) \\ &\text{period}[p,q,h] = p \cdot (m \cdot o) + q \cdot o + h + 2 \end{aligned} \tag{38}$$

Description: $$N_p(k) = \sum_{q=1}^{m}\sum_{h=1}^{o} P_{p,q,h} \cdot b_{p,q,h}(k) \cdot x_q^p[k] \quad (p = 1, \ 2, \ \cdots, \ m) \tag{39}$$

Update for **P** tensor: For each row *p*, solve: $\mathbf{U}_p \cdot \mathbf{p}_p = \mathbf{V}_p$ (40)

$$\begin{aligned} &\mathbf{p}_p \in R^{m \times o}, \quad \mathbf{U}_p \in R^{(m \cdot o) \times (m \cdot o)}, \quad \mathbf{V}_p \in R^{m \cdot o} \\ \text{where: } &U_p[q,q'] = \sum\nolimits_{j,k} x_j^q[k] \cdot b_{p,q,h}(k) \cdot x_j^{q'}[k] \cdot b_{p,q',h'}(k) \\ &V_p[q] = \sum\nolimits_{j,k} t_j^p \cdot x_j^q[k] \cdot b_{p,q,h}(k) \end{aligned} \tag{41}$$

Update for **M** map: For each character i, solve: $\mathbf{W}_i \cdot \mathbf{x}_i = \mathbf{Y}_i$ (42)

$$\mathbf{x}_i \in R^m, \quad \mathbf{W}_i \in R^{m\times m}, \quad \mathbf{Y}_i \in R^m$$

$$\text{where: } W_i[q,q'] = \sum_{j,k}\sum_p \left(\sum_h P_{p,q,h}\cdot b_{p,q,h}(k)\right)\cdot\left(\sum_h P_{p,q',h}\cdot b_{p,q',h}(k)\right) \tag{43}$$
$$Y_i[q] = \sum_{j,k}\sum_p t_j^p \cdot \left(\sum_h P_{p,q,h}\cdot b_{p,q,h}(k)\right)$$

Gradient for **P** tensor: $\dfrac{\partial D}{\partial P_{p,q,h}} = \dfrac{2}{N}\sum_{j,k}\left(N_j^p(k) - t_j^p\right)\cdot x_j^q[k]\cdot b_{p,q,h}(k)$ (44)

Gradient for **M** map: $\dfrac{\partial D}{\partial x_i^q} = \dfrac{2}{N}\sum_{j,k}\sum_p \left(N_j^p(k) - t_j^p\right)\cdot\left(\sum_h P_{p,q,h}\cdot b_{p,q,h}(k)\right)$ (45)

Form 2: P matrix form: In this form, PWFM is implemented as a 2D matrix $\mathbf{P}\in R^{m\times m}$ and each element in the matrix is a coefficient of a basis function $b(k)$. **P** is randomly initialized and trainable. In this case, there are totally $n\times m + m\times m = m(n+m)$ trainable parameters, where n and m have the same meaning as in **Form 1**. Indeed, **P** matrix form is the 2D tensor form. See also: **base**/DD/DDv/ DDvPM.py.

M map: $M: C \rightarrow \mathbf{X}$, a learnable matrix $\mathbf{M}\in R^{m\times n}$, randomly initialized

P matrix: a learnable 2D matrix $\mathbf{P}\in R^{m\times m}$ for the coefficients of basis

$$\text{Basis: } b_{p,q}(k) = \cos\left(\frac{2\pi k}{\text{period}[p,q]}\right) \quad (0 \le p,q < m) \tag{46}$$
$$\text{period}[p,q] = p\cdot m + q + 2$$

$$\text{Description: } N_p(k) = \sum_{q=1}^{m} P_{p,q}\cdot b_{p,q}(k)\cdot x_q[k] \quad (p = 1,\ 2,\ \cdots,\ m) \tag{47}$$

Update for **P** matrix: For each row p, solve: $\mathbf{U}_p \cdot \mathbf{p}_p = \mathbf{V}_p$ (48)

$$\mathbf{p}_p \in R^m, \quad \mathbf{U}_p \in R^{m\times m}, \quad \mathbf{V}_p \in R^m$$

$$\text{where: } U_p[q,q'] = \sum_{j,k} x_j^q[k]\cdot b_{p,q}(k)\cdot x_j^{q'}[k]\cdot b_{p,q'}(k) \tag{49}$$
$$V_p[q] = \sum_{j,k} t_j^p \cdot x_j^q[k]\cdot b_{p,q}(k)$$

Update for **M** map: For each character i, solve: $\mathbf{W}_i \cdot \mathbf{x}_i = \mathbf{Y}_i$ (50)

$$\mathbf{x}_i \in R^m, \quad \mathbf{W}_i \in R^{m\times m}, \quad \mathbf{Y}_i \in R^m$$

$$\text{where: } W_i[q,q'] = \sum_{j,k}\sum_p \left(P_{p,q}\cdot b_{p,q}(k)\right)\cdot\left(P_{p,q'}\cdot b_{p,q'}(k)\right) \tag{51}$$
$$Y_i[q] = \sum_{j,k}\sum_p t_j^p \cdot \left(P_{p,q}\cdot b_{p,q}(k)\right)$$

Gradient for **P** matrix: $\frac{\partial D}{\partial P_{p,q}} = \frac{2}{N}\sum_{j,k}\left(N_j^p(k) - t_j^p\right)\cdot x_j^q[k]\cdot b_{p,q}(k)$ (52)

Gradient for **M** map: $\frac{\partial D}{\partial x_i^q} = \frac{2}{N}\sum_{j,k}\sum_p\left(N_j^p(k) - t_j^p\right)\cdot P_{p,q}\cdot b_{p,q}(k)$ (53)

Form 3: AB matrix form: In this form, PWFM is implemented as a product of two matrices $\mathbf{A}\in R^{m\times L}$ and $\mathbf{B}\in R^{L\times m}$, where **A** is the coefficient matrix and **B** is the basis weight matrix and *L* is the length of the character sequence to be described. For multiple sequences, *L* can be set as the average/maximum length among sequences. **A** is randomly initialized. **B** is initialized by computing basis function weights. For example, the elements in **B** can be calculated as $B[k,p] = b_p(k) = \cos(2\pi\cdot k/p)$ where *k* is row index (*k* = 1, ..., *L*) and *p* is the column index (*p* = 1, …, *m*). In this case, there are totally *m*(*n* + *L*) parameters, where *n* and *m* have the same meaning as in **Form 1** and *L* is the basis length. Indeed, *L* is adjustable and can be regarded as a hidden dimension. By cyclic using vectors in the **A**, **B** matrices via modulus operation, position dependent information is reflected. See also: **base**/DD/DDv/ DDvAB.py.

M map: $M: C\to\mathbf{X}$, a learnable matrix $\mathbf{M}\in R^{m\times n}$, randomly initialized

Acoeff: learnable coefficient matrix $\mathbf{A}\in R^{m\times L}$, randomly initialized

Bbasis: fixed basis matrix $\mathbf{B}\in R^{L\times m}$, where e.g., $B[k,p] = b_p(k) = \cos(2\pi\cdot k/p)$ (54)

Let: *g* = *k* mod *L*, where *L* is the basis length and *g* is the basis index used

for cycling on basis vectors

Description: $\mathbf{N}(k) = \mathbf{A}[:,g]\cdot\mathbf{B}[g,:]\cdot\mathbf{x}[k]$ (55)

Update for **A** matrix: $\mathbf{A}[:,g] = \frac{\sum_{j,k}\mathbf{t}_j\cdot\mathbf{B}[g,:]\cdot\mathbf{x}_j[k]}{\sum_{j,k}\left(\mathbf{B}[g,:]\cdot\mathbf{x}_j[k]\right)^2}$ (56)

Update for **M** map: For each character *i*, solve: $\mathbf{W}_i\cdot\mathbf{x}_i = \mathbf{Y}_i$ (57)

$$\text{where:}\quad \begin{aligned}\mathbf{W}_i &= \sum_{j,k}\|\mathbf{A}[:,g]\|^2\cdot\left(\mathbf{B}[g,:]\otimes\mathbf{B}[g,:]\right)\\ \mathbf{Y}_i &= \sum_{j,k}\left(\mathbf{t}_j\cdot\mathbf{A}[:,g]\right)\cdot\mathbf{B}[g,:]^T\end{aligned} \qquad (58)$$

Gradient for **A** matrix: $\dfrac{\partial D}{\partial A_{p,g}} = \dfrac{2}{m \cdot N} \sum_{j,k} \left(N_j^p(k) - t_j^p \right) \cdot \left(\sum_q B[g,q] \cdot x_j^q[k] \right)$ (59)

Gradient for **M** map: $\dfrac{\partial D}{\partial x_i^q} = \dfrac{2}{m \cdot N} \sum_{j,k} \sum_p \left(N_j^p(k) - t_j^p \right) \cdot A[p,g] \cdot B[g,q]$ (60)

Form 4: Random AB matrix form: In this form, PWFM is implemented as a product of two matrixes $\mathbf{A} \in R^{m \times L}$ and $\mathbf{B} \in R^{L \times m}$, and both **A** and **B** are randomly initialized and trainable. In this case, there are totally $m(n + 2L)$ parameters where n, m, L have the same meaning as in **Form 3**. See also: **base**/DD/DDv/DDvRN.py.

M map: $M : C \to \mathbf{X}$, a learnable matrix $\mathbf{M} \in R^{m \times n}$, randomly initialized

Acoeff: learnable coefficient matrix $\mathbf{A} \in R^{m \times L}$, randomly initialized

Bbasis: learnable basis matrix $\mathbf{B} \in R^{L \times m}$, randomly initialized

Let: $g = k \bmod L$, where L is the basis length and g is the basis index used

for cycling on basis vectors

Description: $\mathbf{N}(k) = \mathbf{A}[:,g] \cdot \mathbf{B}[g,:] \cdot \mathbf{x}[k]$ (61)

Update for **A** matrix: $\mathbf{A}[:,g] = \dfrac{\sum_{j,k} \mathbf{t}_j \cdot \mathbf{B}[g,:] \cdot \mathbf{x}_j[k]}{\sum_{j,k} \left(\mathbf{B}[g,:] \cdot \mathbf{x}_j[k] \right)^2}$ (62)

Update for **B** matrix: For each row g, solve: $\mathbf{U}_g \cdot \mathbf{b}_g = \mathbf{V}_g$ (63)

where: $$\begin{aligned} \mathbf{U}_g &= \sum_{j,k} \left\| \mathbf{A}[:,g] \right\|^2 \cdot \left(\mathbf{x}_j[k] \cdot \mathbf{x}_j[k]^T \right) \\ \mathbf{V}_g &= \sum_{j,k} \left(\mathbf{t}_j \cdot \mathbf{A}[:,g] \right) \cdot \mathbf{x}_j[k] \end{aligned} \quad (64)$$

Update for **M** map: For each character i, solve: $\mathbf{W}_i \cdot \mathbf{x}_i = \mathbf{Y}_i$ (65)

where: $$\begin{aligned} \mathbf{W}_i &= \sum_{j,k} \left\| \mathbf{A}[:,g] \right\|^2 \cdot \left(\mathbf{B}[g,:] \otimes \mathbf{B}[g,:] \right) \\ \mathbf{Y}_i &= \sum_{j,k} \left(\mathbf{t}_j \cdot \mathbf{A}[:,g] \right) \cdot \mathbf{B}[g,:]^T \end{aligned} \quad (66)$$

Gradient for **A** matrix: $\dfrac{\partial D}{\partial A_{p,g}} = \dfrac{2}{m \cdot N} \sum_{j,k} \left(N_j^p(k) - t_j^p \right) \cdot \left(\sum_q B[g,q] \cdot x_j^q[k] \right)$ (67)

Gradient for **B** matrix: $\dfrac{\partial D}{\partial B_{g,q}} = \dfrac{2}{m \cdot N} \sum_{j,k} \sum_{p} \left((N_j^p(k) - t_j^p) \cdot A[p,g] \right) \cdot x_j^q[k]$ (68)

Gradient for **M** map: $\dfrac{\partial D}{\partial x_i^q} = \dfrac{2}{m \cdot N} \sum_{j,k} \sum_{p} \left((N_j^p(k) - t_j^p) \cdot A[p,g] \right) \cdot B[g,q]$ (69)

2.4. Rank definition of DD

Now consider the set formed by double-letter permutations from the character set *C*, that is, the second direct power of *C*: $C \times C = C^2$, where "$\times$" represents the Cartesian product of sets. Since *C* has *n* elements (characters), C^2 has n^2 elements, all of which are (concatenated) strings of length 2, so $C^2 \subset C^*$. These n^2 elements can be regarded as n^2 new characters. The DD of rank-2 refers to the DD formed by the composition weight factors corresponding to the elements in C^2 and the introduced corresponding position weight functions. Generally, for the set formed by *r*-letter permutations (*r*-pers) from the character set *C*, that is, the *r*-th direct power of *C*: $\overset{r}{\underset{i=1}{\times}} C_i = C^n$, the DD of rank-*r* can be defined. The pattern description function and pattern deviation function of high-rank DD can be defined in the same way as for the DD of rank-1 (corresponding to letters in character set *C*). The keys in the CWM of high-rank DDs are called *r*-pers (permutations of length *r*), semantically identical to *k*-mers that are widely acknowledged in the context of biological sequence analysis. High-rank DDs consider the local associations within the character sequence, increasing the number of CWFs.

2.5. Description modes of high-rank DD

When describing a character sequence using high-rank DD, there are two description methods depending on how the subsequences are extracted: "linear" description and "nonlinear" description.

Linear description mode: Linear description uses a technique called the "sliding window". Moving from left to right, it slides one character each time. Thus, linear description has the highest information redundancy.

Nonlinear description mode: Using a tiling (or covering) approach, characters already used are not reused. Thus, it takes every *r* (rank) characters each time in the description. The nonlinear description does not reuse characters from the original sequence; thus, the description is non-redundant. If the sequence length is not exactly divisible by rank, 'padding' or 'dropping' options are needed.

Customized description mode: In addition to the above two extreme modes, there can be something in between, which are called user customized description mode. In this mode, the moving step is between 1 and rank, i.e., 1 < step_size < rank.

2.6. Combined use of multiple DDs

Multiple DDs refer to a group of DDs used jointly to describe sequence characteristics. These descriptors can be of the same rank or different ranks. Each descriptor describes a part of the sequence's characteristics; together they provide a relatively complete description of the sequence.

2.7. ***Numeric DD***

To directly deal with real number/vector sequences, the composition part (map *M*) of DD can be generalized into a transformation matrix $\mathbf{M}^{m\times m}$, and the mapping operation from a character to a *m*-dimensional vector can be generalized as a transformation of the input *m*-dimensional vector via left-multiplying the vector by **M**. The permutation part *P* of DD remains the same as in the vector form of DD. Correspondingly, there are four implementation forms of numeric DD (nDD). The Python implementation of numeric DD classes are presented at: **base**/DD/nDD/.

Form 1: Tensor form: There are totally $m\times m+m\times m\times o=m\times m(1+o)$ trainable parameters in this form, where *m* is the dimension of input vector, and *o* is the number of basis in each expanded position weight function. see also: **base**/DD/nDD/ts/.

M matrix: a learnable transformation matrix $\mathbf{M}\in R^{m\times m}$, randomly initialized

Transformation: $\mathbf{x}[k]=\mathbf{M}\cdot\mathbf{v}[k]\quad(k=1,\ 2,\ \cdots,\ L)$, and **v**[k] is input vector

P tensor: a learnable 3D tensor $\mathbf{P}\in R^{m\times m\times o}$ for the coefficients of basis

Basis: $$b_{p,q,h}(k)=\cos\left(\frac{2\pi k}{\text{period}[p,q,h]}\right) \quad (0\le p,q<m,\ \ 0\le h<o) \tag{70}$$
$$\text{period}[p,q,h]=p\cdot(m\cdot o)+q\cdot o+h+2$$

Description: $$N_p(k)=\sum_{q=1}^{m}\sum_{h=1}^{o}P_{p,q,h}\cdot b_{p,q,h}(k)\cdot x_q^p[k] \quad (p=1,\ 2,\ \cdots,\ m) \tag{71}$$

Update for **P** tensor: For each row p, solve: $\mathbf{U}_p\cdot\mathbf{p}_p=\mathbf{V}_p$ (72)

$$\mathbf{p}_p\in R^{m\times o},\quad \mathbf{U}_p\in R^{(m\cdot o)\times(m\cdot o)},\quad \mathbf{V}_p\in R^{m\cdot o}$$
where: $$U_p[q,q']=\sum\nolimits_{j,k}x_j^q[k]\cdot b_{p,q,h}(k)\cdot x_j^{q'}[k]\cdot b_{p,q',h'}(k) \tag{73}$$
$$V_p[q]=\sum\nolimits_{j,k}t_j^p\cdot x_j^q[k]\cdot b_{p,q,h}(k)$$

Update for **M** matrix: let the flattened vector of **M** as **m**, solve: $\mathbf{W}\cdot\mathbf{m}=\mathbf{Y}$ (74)

$$\mathbf{m}\in R^{m^2},\quad \mathbf{W}\in R^{m^2\times m^2},\quad \mathbf{Y}\in R^{m^2}$$
where: $$W[q,q']=\sum\nolimits_{j,k}\sum\nolimits_p\left(\sum\nolimits_h P_{p,q,h}\cdot b_{p,q,h}(k)\right)\cdot\left(\sum\nolimits_h P_{p,q',h}\cdot b_{p,q',h}(k)\right)\cdot v_j^q(k)\cdot v_j^{q'}(k) \tag{75}$$
$$Y[q]=\sum\nolimits_{j,k}\sum\nolimits_p t_j^p\cdot\left(\sum\nolimits_h P_{p,q,h}\cdot b_{p,q,h}(k)\right)\cdot v_j^q(k)$$

Gradient for **P** tensor: $$\frac{\partial D}{\partial P_{p,q,h}}=\frac{2}{N}\sum_{j,k}\left(N_j^p(k)-t_j^p\right)\cdot x_j^q[k]\cdot b_{p,q,h}(k) \tag{76}$$

Gradient for **M** matrix: $$\frac{\partial D}{\partial M_{q,q'}}=\frac{2}{N}\sum_{j,k}\sum\nolimits_p\left(N_j^p(k)-t_j^p\right)\cdot\left(\sum\nolimits_h P_{p,q,h}\cdot b_{p,q,h}(k)\right)\cdot v_j^{q'}(k) \tag{77}$$

Form 2: P matrix form: There are totally $m\times m+m\times m=2\cdot m\times m$ trainable parameters, where m is the dimension of input vector. see also: **base**/DD/nDD/pm/.

M matrix: a learnable transformation matrix $\mathbf{M}\in R^{m\times m}$, randomly initialized

Transformation: $\mathbf{x}[k]=\mathbf{M}\cdot\mathbf{v}[k]$ $(k=1,\ 2,\ \cdots,\ L)$, and **v**[k] is input vector

P matrix: a learnable 2D matrix $\mathbf{P}\in R^{m\times m}$ for the coefficients of basis

Basis: $$b_{p,q}(k)=\cos\left(\frac{2\pi k}{\text{period}[p,q]}\right) \quad (0\le p,q<m) \tag{78}$$
$$\text{period}[p,q]=p\cdot m+q+2$$

Description: $$N_p(k)=\sum_{q=1}^{m}P_{p,q}\cdot b_{p,q}(k)\cdot x_q^p[k] \quad (p=1,\ 2,\ \cdots,\ m) \tag{79}$$

Update for **P** matrix: For each row p, solve: $\mathbf{U}_p\cdot\mathbf{p}_p=\mathbf{V}_p$ (80)

$$\mathbf{p}_p \in R^m, \quad \mathbf{U}_p \in R^{m\times m}, \quad \mathbf{V}_p \in R^m$$
$$\text{where: } U_p[q,q'] = \sum\nolimits_{j,k} x_j^q[k]\cdot b_{p,q}(k)\cdot x_j^{q'}[k]\cdot b_{p,q'}(k) \tag{81}$$
$$V_p[q] = \sum\nolimits_{j,k} t_j^p \cdot x_j^q[k]\cdot b_{p,q}(k)$$

Update for **M** matrix: let the flattened vector of **M** as **m**, solve: $\mathbf{W}\cdot\mathbf{m} = \mathbf{Y}$ (82)

$$\mathbf{m} \in R^{m^2}, \quad \mathbf{W} \in R^{m^2\times m^2}, \quad \mathbf{Y} \in R^{m^2}$$
$$\text{where: } W[q,q'] = \sum\nolimits_{j,k}\sum\nolimits_p \left(P_{p,q}\cdot b_{p,q}(k)\right)\cdot\left(P_{p,q'}\cdot b_{p,q'}(k)\right)\cdot v_j^q(k)\cdot v_j^{q'}(k) \tag{83}$$
$$Y[q] = \sum\nolimits_{j,k}\sum\nolimits_p t_j^p\cdot\left(P_{p,q}\cdot b_{p,q}(k)\right)\cdot v_j^q(k)$$

Gradient for **P** tensor: $\dfrac{\partial D}{\partial P_{p,q}} = \dfrac{2}{N}\sum\limits_{j,k}\left(N_j^p(k) - t_j^p\right)\cdot x_j^q[k]\cdot b_{p,q}(k)$ (84)

Gradient for **M** matrix: $\dfrac{\partial D}{\partial M_{q,q'}} = \dfrac{2}{N}\sum\limits_{j,k}\sum\nolimits_p\left(N_j^p(k) - t_j^p\right)\cdot P_{p,q}\cdot b_{p,q}(k)\cdot v_j^{q'}(k)$ (85)

Form 3: AB matrix form: There are totally *m*(*m* + *L*) parameters, where *m* is the dimension of input vector and *L* is the basis length. see also: **base**/DD/nDD/ab/.

M matrix: a learnable transformation matrix $\mathbf{M} \in R^{m\times m}$, randomly initialized

Transformation: $\mathbf{x}[k] = \mathbf{M}\cdot\mathbf{v}[k] \quad (k = 1,\ 2,\ \cdots,\ L)$, and **v**[k] is input vector

Acoeff: learnable coefficient matrix $\mathbf{A} \in R^{m\times L}$, randomly initialized

Bbasis: fixed basis matrix $\mathbf{B} \in R^{L\times m}$, where $B[k,p] = b_p(k) = \cos(2\pi\cdot k/p)$ (86)

Let: *g* = *k* mod *L*, where *L* is the basis length and *g* is the basis index used

for cycling on basis vectors

Description: $\mathbf{N}(k) = \mathbf{A}[:,g]\cdot\mathbf{B}[g,:]\cdot\mathbf{x}[k]$ (87)

Update for **A** matrix: $\mathbf{A}[:,g] = \dfrac{\sum\limits_{j,k}\mathbf{t}_j\cdot\mathbf{B}[g,:]\cdot\mathbf{x}_j[k]}{\sum\limits_{j,k}\left(\mathbf{B}[g,:]\cdot\mathbf{x}_j[k]\right)^2}$ (88)

Update for **M** matrix: let the flattened vector of **M** as **m**, solve: $\mathbf{W}\cdot\mathbf{m} = \mathbf{Y}$ (89)

$$\mathbf{m} \in R^{m^2}, \quad \mathbf{W} \in R^{m^2\times m^2}, \quad \mathbf{Y} \in R^{m^2}$$
$$\text{where: } \mathbf{W} = \sum_{j,k}\left\|\mathbf{A}[:,g]\right\|^2\cdot\left(\mathbf{x}[k]\otimes\mathbf{B}[g,:]\right)\otimes\left(\mathbf{x}[k]\otimes\mathbf{B}[g,:]\right) \tag{90}$$
$$\mathbf{Y} = \sum_{j,k}\left(\mathbf{t}_j\cdot\mathbf{A}[:,g]\right)\cdot\operatorname{vec}\left(\mathbf{x}[k]\otimes\mathbf{B}[g,:]\right)$$

Gradient for **A** matrix: $\dfrac{\partial D}{\partial A_{p,g}} = \dfrac{2}{N}\sum_{j,k}\left(N_j^p(k) - t_j^p\right)\cdot\left(\sum_q B[g,q]\cdot x_j^q[k]\right)$ (91)

Gradient for **M** matrix: $\dfrac{\partial D}{\partial M_{q,q'}} = \dfrac{2}{N}\sum_{j,k}\sum_p\left(N_j^p(k) - t_j^p\right)\cdot A[p,g]\cdot B[g,q]\cdot v_j^{q'}[k]$ (92)

Form 4: Random AB matrix form: There are totally $m(m + 2L)$ parameters where m is the dimension of input vector and L is the basis length. see also: **base**/DD/nDD/rn/.

M matrix: a learnable transformation matrix $\mathbf{M}\in R^{m\times m}$, randomly initialized

Transformation: $\mathbf{x}[k] = \mathbf{M}\cdot\mathbf{v}[k] \quad (k = 1,\ 2,\ \cdots,\ L)$, and **v**[k] is input vector

Acoeff: learnable coefficient matrix $\mathbf{A}\in R^{m\times L}$, randomly initialized

Bbasis: learnable basis matrix $\mathbf{B}\in R^{L\times m}$, randomly initialized

Let: $g = k$ mod L, where L is the basis length and g is the basis index used

for cycling on basis vectors

Description: $\mathbf{N}(k) = \mathbf{A}[:,g]\cdot\mathbf{B}[g,:]\cdot\mathbf{x}[k]$ (93)

Update for **A** matrix: $\mathbf{A}[:,g] = \dfrac{\sum_{j,k}\mathbf{t}_j\cdot\mathbf{B}[g,:]\cdot\mathbf{x}_j[k]}{\sum_{j,k}\left(\mathbf{B}[g,:]\cdot\mathbf{x}_j[k]\right)^2}$ (94)

Update for **B** matrix: For each row g, solve: $\mathbf{U}_g\cdot\mathbf{b}_g = \mathbf{V}_g$ (95)

$$\mathbf{b}_g \in R^m,\quad \mathbf{U}_g \in R^{m\times m},\quad \mathbf{V}_g \in R^m$$

where: $\mathbf{U}_g = \sum_{j,k}\|\mathbf{A}[:,g]\|^2\cdot\left(\mathbf{x}_j[k]\cdot\mathbf{x}_j[k]^T\right)$ (96)

$$\mathbf{V}_g = \sum_{j,k}\left(\mathbf{t}_j\cdot\mathbf{A}[:,g]\right)\cdot\mathbf{x}_j[k]$$

Update for **M** matrix: let the flattened vector of **M** as **m**, solve: $\mathbf{W}\cdot\mathbf{m} = \mathbf{Y}$ (97)

$$\mathbf{m}\in R^{m^2},\quad \mathbf{W}\in R^{m^2\times m^2},\quad \mathbf{Y}\in R^{m^2}$$

where: $\mathbf{W} = \sum_{j,k}\|\mathbf{A}[:,g]\|^2\cdot\left(\mathbf{x}[k]\otimes\mathbf{B}[g,:]\right)\otimes\left(\mathbf{x}[k]\otimes\mathbf{B}[g,:]\right)$ (98)

$$\mathbf{Y} = \sum_{j,k}\left(\mathbf{t}_j\cdot\mathbf{A}[:,g]\right)\cdot\mathrm{vec}\left(\mathbf{x}[k]\otimes\mathbf{B}[g,:]\right)$$

Gradient for **A** matrix: $\dfrac{\partial D}{\partial A_{p,g}} = \dfrac{2}{N}\sum_{j,k}\left(N_j^p(k) - t_j^p\right)\cdot\left(\sum_q B[g,q]\cdot x_j^q[k]\right)$ (99)

Gradient for **B** matrix: $\frac{\partial D}{\partial B_{g,q}} = \frac{2}{N}\sum_{j,k}\sum\nolimits_{p}\left((N_j^p(k)-t_j^p)\cdot A[p,g]\right)\cdot x_j^q[k]$ (100)

Gradient for **M** matrix: $\frac{\partial D}{\partial M_{q,q'}} = \frac{2}{N}\sum_{j,k}\sum\nolimits_{p}\left(N_j^p(k)-t_j^p\right)\cdot A[p,g]\cdot B[g,q]\cdot v_j^{q'}[k]$ (101)

2.8 Hierarchical DD

As mentioned in Ma (2003), DD can be used recursively to give a hierarchical description of character sequence. Based on the numeric DD, hierarchical DD (hDD) can be defined which recursively use numeric DD to transform the input vector sequence into different dimension or length (via Linker matrix). In hDD, each layer is a numeric vector DD. hDD can be trained by using gradient descent method. To support deeper layers, tricks from modern deep-learning methods such as pre-layer normalization and residual connection can be adopted. The Python implementation of hierarchical DD classes are presented at: **base**/DD/hDD/.

2.8.1 Forward Computation

Suppose the layer index is *l*, the forward computation of hDD is:

$$\begin{aligned} &\mathbf{N}^{(l)}(k) = \mathbf{F}^{(l)}\left(\mathbf{M}^{(l)}\cdot\mathbf{N}^{(l-1)}(k)\right) = \mathbf{P}^{(l)}(k)\cdot\mathbf{M}^{(l)}\cdot\mathbf{N}^{(l-1)}(k) \quad \text{for} \quad l \geq 1 \\ &\mathbf{N}^{(0)} = \mathbf{v}(k) \quad \text{for} \quad l = 0 \end{aligned} \tag{102}$$

where $\mathbf{M}^{(l)} \in R^{m^{(l)}\times m^{(l-1)}}$ is the transformation matrix for layer *l*, and $\mathbf{F}^{(l)} \in R^{m^{(l)}}$ is a vector function that depends on the PWFM of layer *l*: $\mathbf{P}^{(l)}(k)$. Corresponding to nDD, there are four implementation forms of $\mathbf{P}^{(l)}(k)$ as Forms 1 – 4, namely, **Tensor**, **P matrix**, **AB matrix** and **Random AB matrix** forms. In the formula, **v**(k) is the input vector sequence; $m^{(l)}$ and $m^{(l-1)}$ are the model dimensions of layer *l* and layer *l*-1, respectively.

2.8.2 Error Backpropagation

Deviation is defined based on the model of last layer *L*:

$$D = \frac{1}{N}\sum\nolimits_{j,k}\left\|\mathbf{N}^{(L)}(k)-\mathbf{t}_j\right\|^2 . \tag{103}$$

The error signal at the output layer (*l* = *L*) is defined as:

$$\delta_p^{(L)}(k) = \frac{\partial D}{\partial N_p^{(L)}(k)} = \frac{2}{N}\left(N_p^{(L)} - t_{j,p}\right) \qquad (p = 1, 2, \cdots, m^{(L)})\,. \tag{104}$$

In vector form:

$$\boldsymbol{\delta}^{(L)}(k) = \frac{2}{N}\left(\mathbf{N}^{(L)}(k) - \mathbf{t}_j\right). \tag{105}$$

For layer $l < L$, the error propagated through the $F^{(l)}(\mathbf{x};k)$ transformation is:

$$\delta_{x_q}^{(l)}(k) = \frac{\partial D}{\partial x_q^{(l)}(k)} = \sum_{p=1}^{m^{(l)}} \delta_p^{(l)}(k) \cdot \frac{\partial [F^{(l)}(\mathbf{x};k)]_p}{\partial x_q} \tag{106}$$

where $F^{(l)}(\mathbf{x};k)$ depends on the PWFM $\mathbf{P}^{(l)}(k)$ of layer l and has four implementation forms as in nDD.

The error propagated through the **M** transformation is:

$$\delta_p^{(l-1)}(k) = \frac{\partial D}{\partial N_p^{(l-1)}(k)} = \sum_{q=1}^{m^{(l)}} \delta_{x_q}^{(l)}(k) \cdot \frac{\partial x_q^{(l)}(k)}{\partial N_p^{(l-1)}(k)} = \sum_{q=1}^{m^{(l)}} \delta_{x_q}^{(l)}(k) \cdot M_{q,p}^{(l)}\,. \tag{107}$$

In matrix form:

$$\boldsymbol{\delta}^{(l-1)}(k) = \left(\mathbf{M}^{(l)}\right)^T \cdot \boldsymbol{\delta}_{\mathbf{x}}^{(l)}(k)\,. \tag{108}$$

2.8.3 Parameter Updates

The gradient with respect to the $\mathbf{P}^{(l)}(k)$ matrix is:

$$\frac{\partial D}{\partial \mathbf{P}^{(l)}(k)} = \frac{1}{N}\sum\nolimits_{j,k} \boldsymbol{\delta}^{(l)}(k) \cdot \mathbf{x}^{(l)}(k) \cdot \mathbf{b}^{(l)}(k)\,. \tag{109}$$

The gradient with respect to the $\mathbf{M}^{(l)}$ matrix is:

$$\frac{\partial D}{\partial \mathbf{M}^{(l)}} = \frac{1}{N}\sum\nolimits_{j,k} \boldsymbol{\delta}^{(l)}(k) \cdot \mathbf{N}^{(l-1)}(k)\,. \tag{110}$$

P and **M** are updated according to:

$$\begin{aligned} \mathbf{P}^{(l)} &\leftarrow \mathbf{P}^{(l)} - \eta \cdot \frac{\partial D}{\partial \mathbf{P}^{(l)}} \\ \mathbf{M}^{(l)} &\leftarrow \mathbf{M}^{(l)} - \eta \cdot \frac{\partial D}{\partial \mathbf{M}^{(l)}} \end{aligned} \tag{111}$$

where η is the learning rate.

2.8.4 Complete Backpropagation Algorithm

The complete backpropagation algorithm can be summarized as follows:

(1) **Forward Pass**: Compute all $\mathbf{N}^{(l)}(k)$ for each layer and position.

(2) **Output Error**: Compute $\boldsymbol{\delta}^{(l)}(k)$ for all positions.

(3) **Backward Pass**: For each layer from *L* down to 1:

(i) Compute gradients for $\mathbf{P}^{(l)}$;

(ii) Compute gradients for $\mathbf{M}^{(l)}$;

(iii) Propagate error to previous layer: $\boldsymbol{\delta}^{(l-1)}(k)=\left(\mathbf{M}^{(l)}\right)^T\cdot\boldsymbol{\delta}^{(l)}(k)$.

(4) **Parameter Update**: Update all parameters using the computed gradients.

2.8.5 hDD with Link matrix

Linker matrix can be imported between hDD layers, and with Linker matrix, the sequence length can be transformed. For each layer *l*, the transformation is:

$$\mathbf{O}^{(l)}=\mathbf{F}^{(l)}\cdot\mathbf{L}^{(l)} \tag{112}$$

where $\mathbf{O}^{(l)}\in R^{m^{(l)}\times s^{(l)}}$ is the output matrix of layer *l* (model dimension × sequence length), $\mathbf{F}^{(l)}\in R^{m^{(l)}\times s^{(l-1)}}$ is the forward computation/transformation matrix of layer *l*, and $\mathbf{L}^{(l)}\in R^{s^{(l-1)}\times s^{(l)}}$ is the Linker matrix for sequence length transformation between layer *l* - 1 and layer *l*.

Deviation is defined based on the model of last layer *L*:

$$D=\frac{1}{N}\sum\nolimits_{j,k}\left\|\mathbf{N}^{(L)}(k)-\mathbf{t}_j\right\|^2 \tag{113}$$

where *N* is the total number of positions in the sequences. The output layer gradient is:

$$\frac{\partial D}{\partial O_{i,j}^{(L)}}=\frac{2}{N\cdot s^{(L)}}\left(O_{i,j}^{(L)}-t_j^i\right). \tag{114}$$

The gradient for Linker matrix of layer *l* is:

$$\frac{\partial D}{\partial L_{p,q}^{(l)}}=\sum_{i=1}^{m^{(l)}}\sum_{k=1}^{s^{(l)}}\frac{\partial D}{\partial O_{i,k}^{(l)}}\cdot\frac{\partial O_{i,k}^{(l)}}{\partial L_{p,q}^{(l)}}=\sum_{i=1}^{m^{(l)}}F_{i,p}^{(l)}\cdot\frac{\partial D}{\partial O_{i,q}^{(l)}}. \tag{115}$$

2.9 DD network

DD network (DDN) is defined as a combination of DD (especially hDD) with other machine learning modules such as multilayer perceptron (MLP), convolutional neural network (CNN), recurrent neural network (RNN), graph neural network (GNN), transformers and so on. DDN enhances modeling ability by combining the capabilities of these modules. Some demo examples of DDN are presented at: **base**/DD/DDN/ and **base**/PyTorchDD/DDN/.

3. Application of DD to sequence problems

3.1 using DD for feature extraction

DD is intrinsically sequence feature extractors. After training, all the parameters in DD like Position Weight Coefficients (PWCs), Composition Weight Factors (CWFs), carry the information from the sequences in the training dataset. Meanwhile, the Position-Weighted Frequencies (PWFs):

$$F_{x_i} = \frac{P_{x_i}}{L} = \frac{1}{L}\sum_{k_{x_i}} P(k_{x_i}) \qquad (x_i \in X) \tag{116}$$

and the Partial Dual Variables (PDVs):

$$V_{x_i} = \frac{1}{L}\sum_{k_{x_i}} P(k_{x_i}) \cdot x_i \qquad (x_i \in X) \tag{117}$$

also reflect sequence information in the training dataset. These parameters can be flattened into feature vectors to be used for classification or regression tasks in other machine learning approaches such as Fisher discrimination (as demonstrated in Ma, 2003 and 2007), support vector machine, random forest, neural networks, transformers, and so on.

3.2 using DD for identification

As demonstrated in Ma (2003), DD can be used for sequence identification like gene identification via ***d*-value threshold method**. This method is the intrinsic method of DD for sequence identification. The training process of DD is minimizing the deviation value (*d*-value). After training, the parameters in DD capture the sequence patterns in the training dataset, which means a sequence with similar patterns will yield small *d*-value when it is described by the trained DD. For a new sequence to be identified, the trained DD can be used to calculate its *d*-value. If the *d*-value of a sequence is smaller than a threshold, this sequence can be identified as the same type of those in the training dataset. The threshold of *d*-value can be determined by setting False Positive = False Negative on the training or verification dataset. The ***d*-value threshold method** can be used in multi-label problems by training multiple DDs and setting multiple thresholds on each label's dataset, respectively. For multi-label problems, same or different targets can be set for different labels in the deviation calculation during training and prediction processes, consistently.

3.3 using DD for classification

As demonstrated in Ma (2008), DD can be used for sequence classification. For two-class problems, DD can be trained on positive (Class P) and negative (class N) samples, respectively, resulting in two trained DDs (DD*p* and DD*n*). For a new sequence to be classified, DD*p* and DD*n* are used to calculated its *d*-value, respectively, yielding two *d*-values (d_p and d_n). The classification of a sequence *s* is just to see which one of d_p and d_n is the smaller. If d_p is less than d_n, then the sequence *s* belongs to Class P (positive sample); otherwise, the sequence *s* belongs to Class N (negative sample). For multi-class problems, train DD for each class, respectively, resulting in multiple trained DDs (DD*i*, where *i* is the Class index). For a new sequence *s* to be classified, using DD*i* to calculate its *d*-value (d_i), the classification of *s* is then determined by the smallest *d*-value: $i = \arg\min(d_i)$. For multi-class problems, same or different targets can be set for different classes in the deviation calculation during training and prediction processes, consistently.

3.4 using DD for regression

DD is intrinsically a regressor. The deviation in DD model is defined as the mean squared error (MSE) between the local description $N(k)$ at each position k and the global target t for a sequence, averaged over all sequences and positions in the training set. By minimizing the deviation at each position, the DD model learns a representation where the local patterns $N(k) = P(k) \cdot x[k]$ collectively encode the global target. For the prediction in regression, the predicting target formula Eq. (22) can be used. After training, predicting the target value for a new sequence is both natural and optimal under the model's own objective. The method computes the $N(k)$ value for every position k in the sequence. The predicted target t_{pred} is then simply the arithmetic mean of these $N(k)$ values. This approach is optimal because it minimizes the MSE between the local description and the predicted target for a specific sequence, making the prediction consistent with the training paradigm. The regression power of DD may also be exploited for classification tasks by setting class-specific discrete targets in training and using argmax/softmax transformations in prediction.

3.5 using DD for generation

DD can be used for sequence generation by learning both compositional and positional patterns from training data. The generation process leverages the trained model parameters to create new sequences that reflect the statistical properties of the original dataset. The core generation mechanism operates as follows. After training, the DD model captures two key aspects of sequence structure: the Composition Weight Map (CWM), which assigns weights to each r-per/k-mer token, and the Position Weight Coefficients (PWC), which define how these weights are modulated across sequence positions. The generation process uses these learned parameters to construct sequences token-by-token, where each token is selected to minimize the deviation between its weighted contribution and the average target value observed during training.

The generation method takes a target length L and an optional temperature parameter τ. For each position k, the score for token selection is defined as:

$$Score(k,\text{tok}) = -\left(P(k)\cdot x_{\text{tok}} - t_{\text{avg}}\right)^2 \tag{118}$$

where t_{avg} is the average target value in the training data. When $\tau = 0$, the method deterministically selects the token that maximizes this score.

$$\text{tok}_{\text{selected}}(k) = \arg\max\left(Score(k,\text{tok})\right) \qquad \text{tok} \in \{\text{valid tokens}\}\,. \tag{119}$$

For $\tau > 0$, a softmax transformation is applied to the scores, introducing stochasticity that increases with τ:

$$\text{Probability}(\text{tok} \mid k) = \frac{\exp\left(Score(k,\text{tok})/\tau\right)}{\sum_{\text{tok}'}\exp\left(Score(k,\text{tok}')/\tau\right)} \qquad \text{tok} \in \{\text{valid tokens}\}\,, \tag{120}$$

thereby enabling controlled exploration of the sequence space. For generation purpose, self-supervised training can be adopted where the target is from the training sequence itself rather than from outside. In the self-supervised training, both “gap filling” and “auto regression” modes are supported.

Availability

An object-oriented implementation of Dual Descriptor in Python/PyTorch that wraps the training and prediction algorithms is freely available at GitHub: https://github.com/mbglab/DualDescriptor.

Reference

Ma, B.G. 2003. Analytic Number Theory Model for character sequence and its application in bioinformatics. Master Thesis, Tian Jing University. Link to CNKI.

Ma, B.G. 2007. How to describe genes: enlightenment from the quaternary number system. Biosystems 90, 20-27.

Ma, B.G. 2008. Building Block and Building Rule: Dual Descriptor Method for Biological Sequence Analysis. Nature Procedings: npre.2008.2223.1.