Learning Computational Verb Rules

Tao Yang

Abstract— **Computational verb rules are efficient tools to transform dynamical experiences in natural languages into mathematical expressions. Based on human experiences, we can easily construct computational verb rules. However, due to the limits of human perceptions, it is impossible to design accurate computational verbs based on human perceptions. To calibrate the parameters of computational verbs in computational verb rules, we need to train them by using measurements that are much more accurate than human perceptions. Different kinds of learning algorithms of computational verb rules, of which the consequents and antecedents are static or dynamical, are presented.** *Copyright © 2007 Yang's Scientific Research Institute, LLC. All rights reserved.*

Index Terms— **Computational verb rules, learning, physical linguistics, minimization.**

I. Introduction

IN PHYSICAL linguistics, the Universe consists of BEING's that are represented as nouns in natural languages. Each noun is associated with attributes and actions. The essential difference between physical linguistics and conventional linguistics is that in the former each noun is not only a symbol but also a measurable and computable entity called *computational noun*. Each computational noun is modeled by its *attribute values* together with its *action values*.

1) An attribute value is a mathematical function defined in some measurable domains that are usually physical in engineering contexts. Some examples of attribute values are: Characteristic functions of sets, membership functions of fuzzy sets, and probability density functions of events.
2) An action value is a mathematical function defined in some measurable domains and time, therefore, an action value is an observation of the dynamics of processes and actions. An action value is closely related to the relations or configurations among computational nouns. Action values are mathematically represented as computational verbs.

Computational verb logic[14], [23] is a logic consisting of computational verbs as its essential building blocks. In engineering applications, we use computational verb rules to represent knowledge and perform the computation among computational verb rules based on principles presented in computational verb logic. Computational verb rules play critical roles in many applications such as computational verb control, computational verb image processing, computational verb database and computational verb knowledge representation.

Coupling with the accurate measurements from different types of sensors, computational verb rules can efficiently root knowledge structures coded in natural language into accurate measurements in numbers. However, since computational verb rules are usually constructed based on human experiences, we need to calibrate the parameters of all computational verbs before applying them to accurate measurements. There are at least the following ways to calibrate the parameters of computational verbs in computational verb rules.

- To apply clustering algorithms to find template computational verbs from historical records as reported in [37].
- To choose standard computational verbs based on knowledge of the underlying system to model. This method is only useful when we have enough knowledge about the underlying system. For example, if we use computational verbs to model lumped models of electronic circuits, the most typical dynamics are already known.
- To learn computational verbs in computational verb rule by training samples.

Here we will study the third method mentioned above.

In general, since computational verbs are dynamical systems, it is extremely difficult to design the learning algorithm in the form of evolving functions. To reduce the difficulty, the technology that will be used is to construct all computational verbs based on a few canonical computational verbs. By doing so, a computational verb can be fully defined by a set of canonical computational verbs and a parameter vector. By using this technology, to choose a computational verb from a function space is simply to choose a parameter vector from a parameter space. Since a parameter space is of finite number of dimensions, the learning problems of computational verb rules are degenerated from infinite dimensions to finite dimensions.

The method presented in this paper can be easily applied to the design of a new type of neural networks called *computational verb neural networks*(VNN) because a VNN structure is an interconnected structure of implementing computational verb rules. The learning algorithm of VNN will be the same as those used to train computational verb rules. The same as the learning algorithm for conventional rules or neural networks, the learning algorithms for computational verb rules will most likely take local minimums as solutions; namely, can only find sub-optimal solutions.

The organization of this paper is as follows. In Section II, the brief history of computational verb theory will be given. In Section III, the learning algorithms for single- and multi-input computational verb rules with real number consequents

Manuscript received May 3, 2007; revised July 23, 2007
Tao Yang, Department of Electronic Engineering, Xiamen University, Xiamen 361005, P. R. China. Department of Cognitive Economics, Department of Electrical Engineering and Computer Sciences, Yang's Scientific Research Institute, 1303 East University Blvd., #20882, Tucson, Arizona 85719-0521, U.S.A. Email: taoyang@xmu.edu.cn, taoyang@yangsky.com, taoyang@yangsky.us.
Publisher Item Identifier S 1542-5908(07)10303-1/$20.00

and computational verb antecedents will be exploited. In Section IV, the problems of learning computational verb rules based on training samples will be recast into solving nonlinear equations in parameter spaces. In Section V, the learning problems will be transformed into finding minimum points of the error functions and some tools will be given with illustrating examples. In Section VI, the learning algorithms for computational verb rules with computational verb consequents and static antecedents will be presented. The cases of real number antecedents and fuzzy set antecedents will be studied. In Section VII, the learning algorithm for computational verbs with computational verb antecedents and consequents will be studied. In Section VIII, some concluding remarks will be presented.

II. A Brief History of Computational Verb Theory

As the first paradigm shift for solving engineering problems by using verbs, the computational verb theory and physical linguistics have undergone a rapid growth since the birth of computational verb in the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley in 1997[9], [10]. The paradigm of implementing verbs in machines were coined as *computational verb theory*[23]. The building blocks of computational theory are *computational verbs*[18], [13], [11], [19], [24]. The relation between verbs and adverbs was mathematically defined in [12]. The logic operations between verb statements were studied in [14]. The applications of verb logic to verb reasoning were addressed in [15] and further studied in [23]. A logic paradox was solved based on verb logic[20]. The mathematical concept of set was generalized into verb set in[17]. Similarly, for measurable attributes, the number systems can be generalized into verb numbers[21]. The applications of computational verbs to predictions were studied in [16]. In [25] fuzzy dynamic systems were used to model a special kind of computational verb that evolves in a fuzzy space. The relation between computational verb theory and traditional linguistics was studied in [23], [26]. The theoretical basis of developing computational cognition from a unified theory of fuzzy and computational verb theories is the theory of the Unicogse that was studied in [26], [31]. The issues of simulating cognition using computational verbs were studied in [27]. A way to implementing feelings in machines was proposed based on grounded computational verbs and computational nouns in [33]. In [40] a new definition of the similarity between computational verbs was studied. The theory of computational verb has been taught in some university classrooms since 2005[1].

The latest active applications of computational verb theory are listed as follows.

1) Computational Verb Controllers. The applications of computational verbs to different kinds of control problems were studied on different occassions[22], [23]. For the advanced applications of computational verbs to control problems, two papers reporting the latest advances had been published[29], [28]. The design of computational verb controller was also presented in a textbook in 2006[1].
2) Computational Verb Image Processing and Image Understanding. The recent results of image processing by using computational verbs can be found in[30]. The applications of computational verbs to image understanding can be found in [32].
3) Stock Market Modeling and Prediction based on computational verbs. The product of Cognitive Stock Charts[4] was based on the advanced modeling and computing reported in [34]. Applications of computational verbs was used to study the trends of stock markets known as Russell reconstruction patterns [35].

Computational verb theory has been successfully applied to many industrial and commercial products. Some of these products are listed as follows.

1) Visual Card Counters. The *YangSky-MAGIC* card counter[6], developed by Yang's Scientific Research Institute and Wuxi Xingcard Technology Co. Ltd., was the first visual card counter to use computational verb image processing technology to achieve high accuracy of card and paper board counting based on cheap webcams.
2) CCTV Automatic Driver Qualify Test System. The *DriveQfy* CCTV automatic driver qualify test system[7] was the first vehicle trajectory reconstruction and stop time measuring system using computational verb image processing technology.
3) Visual Flame Detecting System. The *FireEye* visual flame detecting system[2] was the first CCTV or webcam based flame detecting system, that works under color and black & white conditions, for surveillance and security monitoring system[38], [39].
4) Smart Pornographic Image and Video Detection Systems. The *PornSeer*[5] pornographic image and video detection systems are the first cognitive feature based smart porno detection and removal software.
5) Webcam Barcode Scanner. The *BarSeer*[3] webcam barcode scanner took advantage of the computational verb image processing to make the scan of barcode by using cheap webcam possible.
6) Cognitive Stock Charts. By applying computational verbs to the modeling of trends and cognitive behaviors of stock trading activities, cognitive stock charts can provide the traders with the "feelings" of stock markets by using simple and intuitive indexes.

III. Learning Computational Verb Rules with Real Number Consequents

The first of two simplest verb rules is of the form

$$\text{IF } x(t) \text{ V, THEN } y \text{ is } c \tag{1}$$

where the antecedent consists of one computational verb and the consequent is a real number. In physical linguistics, $x(t)$, the waveform is the action value of a computational noun, e.g.,

[1]Dr. G. Chen, EE 64152 - Introduction to Fuzzy Informatics and Intelligent Systems, Department of Electronic Engineering, City University of Hong Kong. Dr. Mahir Sabra, EELE 6306: Intelligent Control, Electrical and Computer Engineering Department, The Islamic University of Gaza.

input, and $c \in \mathbb{R}$ is the attribute value of computational noun y. The second of two simplest verb rule is of the form

$$\text{IF } x \text{ is } c \text{ , THEN } y(t) \text{ V} \tag{2}$$

where the antecedent is a real number and the consequent consists of one computational verb. In physical linguistics, $c \in \mathbb{R}$ is the attribute values of computational noun x and $y(t)$, the waveform is the action values of a computational noun, e.g., output. The difference between computational verb rules (1) and (2) is that the former has static consequent while the latter has static antecedent.

If all computational verbs in the antecedent and consequent of a computational rule degenerate into be's, then the computational verb rule degenerates into a conventional rule.Some examples of computational verb rules are listed as follow.

- IF temperature becomes `high`, THEN current is `off`.
- IF speed decreases, THEN switch is `on`.
- IF the microwave is `on`, THEN water becomes `hot`.
- IF voltage is `3.3`V, THEN current oscillates.

Here we study the learning algorithm of computational verb rule (1).

A. Single-input Computational Verb Algorithms

Let's assume the following *computational verb algorithm* of n computational verb rules.

$$\begin{aligned}&\text{IF } x(t) \text{ V}_1\text{, THEN } y \text{ is } y_1;\\ &\vdots\\ &\text{IF } x(t) \text{ V}_n\text{, THEN } y \text{ is } y_n,\end{aligned} \tag{3}$$

where $x(t) \in \mathbb{R}, t \in [0, T]$ is a measured waveform known as *input*, and y is the *output*, of which the attribute values are real numbers $y_i \in \mathbb{R}$. $\text{V}_i, i = 1, \dots, n$, are n computational verbs of which the evolving functions are $\mathcal{E}_i(t) \in \mathbb{R}, t \in [0, T], i = 1, \dots, n$. Observe that each computational verb rule in algorithm (3) consists of an antecedent of one computational verb and a consequent of a real number.

Given a computational verb similarity $S(\cdot, \cdot)$, the output of the entire computational verb algorithm is given by

$$y = \frac{\displaystyle\sum_{i=1}^{n} S(x(t), \mathcal{E}_i(t)) y_i}{\displaystyle\sum_{i=1}^{n} S(x(t), \mathcal{E}_i(t))}. \tag{4}$$

The computational verb similarity is one of the most important concept in computational verb logic and has been studied in very detained in [36]. Here we use the following computational verb similarity as an illustrating example

$$S(\mathcal{E}_1(t), \mathcal{E}_2(t)) = 1 - \sqrt{\frac{1}{T}\int_0^T [s_t(\mathcal{E}_1(t)) - s_t(\mathcal{E}_2(t))]^2 \, dt} \tag{5}$$

where $s_t(\cdot)$ is a *saturate function*[36] given by

$$s_t(x) = \frac{1}{1 + e^{-x}}, \text{ and } \dot{s}_t(x) = \frac{e^{-x}}{(1 + e^{-x})^2}. \tag{6}$$

We will back to this computational verb similarity later when we apply it to derive the learning algorithms of computational verb rules.

We assume that there are m canonical computational verbs $\widetilde{\mathcal{E}}_j(t), j = 1, \dots, m$, which are used to construct all computational verbs as

$$\mathcal{E}_i(t) = \sum_{j=1}^{m} \alpha_{ij} \widetilde{\mathcal{E}}_j(t) \tag{7}$$

where $\alpha_{ij} \in \mathbb{R}, i = 1, \dots, n; j = 1, \dots, m$, function as adverbs.

Given K sets of training samples $\{(u_k(t), d_k)\}_{k=1}^{K}$, we define an error function as

$$E = \sum_{k=1}^{K} (\tilde{y}_k - d_k)^2 \tag{8}$$

where

$$\tilde{y}_k = \frac{\displaystyle\sum_{i=1}^{n} S(u_k(t), \mathcal{E}_i(t)) y_i}{\displaystyle\sum_{i=1}^{n} S(u_k(t), \mathcal{E}_i(t))}. \tag{9}$$

The learning rule is given by

$$\alpha_{ij}(l + 1) = \alpha_{ij}(l) + \gamma(l)\Delta\alpha_{ij}(l), \quad l = 1, \dots, \tag{10}$$

where $\gamma(l) \in \mathbb{R}^+$ is the learning rate, which can be tuned to different values during the training process, and

$$\begin{aligned}\Delta\alpha_{ij} &= -\frac{\partial E}{\partial \alpha_{ij}}\\ &= -\frac{\partial \displaystyle\sum_{k=1}^{K} (\tilde{y}_k - d_k)^2}{\partial \alpha_{ij}}\\ &= -2\sum_{k=1}^{K} (\tilde{y}_k - d_k)\frac{\partial \tilde{y}_k}{\partial \alpha_{ij}}\end{aligned} \tag{11}$$

where

$$\begin{aligned}\frac{\partial \tilde{y}_k}{\partial \alpha_{ij}} &= \frac{\partial \dfrac{\sum_{i=1}^{n} S(u_k(t), \mathcal{E}_i(t)) y_i}{\sum_{i=1}^{n} S(u_k(t), \mathcal{E}_i(t))}}{\partial \alpha_{ij}}\\ &= \frac{1}{\displaystyle\sum_{i=0}^{n} S(u_k(t), \mathcal{E}_i(t))} \frac{\partial S(u_k(t), \mathcal{E}_i(t)) y_i}{\partial \alpha_{ij}}\\ &\quad - \frac{\displaystyle\sum_{i=0}^{n} S(u_k(t), \mathcal{E}_i(t)) y_i}{\left[\displaystyle\sum_{i=0}^{n} S(u_k(t), \mathcal{E}_i(t))\right]^2} \frac{\partial S(u_k(t), \mathcal{E}_i(t))}{\partial \alpha_{ij}}\end{aligned} \tag{12}$$

where $\frac{\partial S(u_k(t), \mathcal{E}_i(t))}{\partial \alpha_{ij}}$ is given by Eq. (13). Observe that in Eq. (13) only one of many possible computational verb similarities is used. We can use other computational verb similarities as well.

$$\begin{aligned}\frac{\partial S(u_k(t),\mathcal{E}_i(t))}{\partial\alpha_{ij}} &= \frac{\partial\left\{1-\sqrt{\frac{1}{T}\int_0^T\left[s_t\left(u_k(t)\right)-s_t\left(\sum_{j=1}^m\alpha_{ij}\widetilde{\mathcal{E}}_j(t)\right)\right]^2 dt}\right\}}{\partial\alpha_{ij}}\\ &= \frac{\int_0^T\left[s_t\left(u_k(t)\right)-s_t\left(\sum_{j=1}^m\alpha_{ij}\widetilde{\mathcal{E}}_j(t)\right)\right]\dot{s}_t\left(\sum_{j=1}^m\alpha_{ij}\widetilde{\mathcal{E}}_j(t)\right)\widetilde{\mathcal{E}}_j(t)dt}{T\sqrt{\frac{1}{T}\int_0^T\left[s_t(u_k(t))-s_t\left(\sum_{j=1}^m\alpha_{ij}\widetilde{\mathcal{E}}_j(t)\right)\right]^2 dt}}. \qquad (13)\end{aligned}$$

B. Multi-input Computational Verb Algorithms

Let's assume the following *computational verb algorithm* of n computational verb rules.

$$\begin{aligned}&\text{IF } x_1(t)\ \mathsf{V}_{11} \text{ AND } x_2(t)\ \mathsf{V}_{12} \ldots \text{AND } x_p(t)\ \mathsf{V}_{1p},\\ &\text{THEN } y \text{ is } y_1;\\ &\vdots\\ &\text{IF } x_1(t)\ \mathsf{V}_{n1} \text{ AND } x_2(t)\ \mathsf{V}_{n2} \ldots \text{AND } x_p(t)\ \mathsf{V}_{np},\\ &\text{THEN } y \text{ is } y_n \qquad (14)\end{aligned}$$

where $x_i(t)\in\mathbb{R}, t\in[0,T], i=1,\ldots,p$ are p inputs. $\mathsf{V}_{ij}, i=1,\ldots,n; j=1,\ldots,p$, are np computational verbs, of which the evolving functions are $\mathcal{E}_{ij}(t)\in\mathbb{R}, t\in[0,T], i=1,\ldots,n; j=1,\ldots,p$. Observe that the antecedent of each computational verb rule consists of p computational verbs while the consequent is a real number.

The output of this multi-input computational algorithm is given by

$$y=\frac{\sum_{i=1}^n y_i\prod_{j=1}^p S(x_j(t),\mathcal{E}_{ij}(t))}{\sum_{i=1}^n\prod_{j=1}^p S(x_j(t),\mathcal{E}_{ij}(t))}. \qquad (15)$$

We assume that there are m canonical computational verbs $\widetilde{\mathcal{E}}_j(t), j=1,\ldots,m$, which are used to construct all computational verbs as

$$\mathcal{E}_{ih}(t)=\sum_{j=1}^m\alpha_{ij}^{(h)}\widetilde{\mathcal{E}}_j(t),\quad h=1,\ldots,p, \qquad (16)$$

where $\alpha_{ij}^{(h)}\in\mathbb{R}, i=1,\ldots,n; j=1,\ldots,m$, function as adverbs.

Given K sets of training samples $\{(u_{k1}(t),\ldots,u_{kp}(t),d_k)\}_{k=1}^K$, we define an error function as

$$E=\sum_{k=1}^K(\tilde{y}_k-d_k)^2 \qquad (17)$$

where

$$\tilde{y}_k=\frac{\sum_{i=1}^n y_i\prod_{h=1}^p S(u_{kh}(t),\mathcal{E}_{ih}(t))}{\sum_{i=1}^n\prod_{h=1}^p S(u_{kh}(t),\mathcal{E}_{ih}(t))}. \qquad (18)$$

The learning rule is given by

$$\alpha_{ij}^{(h)}(l+1)=\alpha_{ij}^{(h)}(l)+\gamma(l)\Delta\alpha_{ij}^{(h)}(l),\quad l=1,\ldots, \qquad (19)$$

where $\gamma(l)\in\mathbb{R}^+$ is the learning rate, which can be tuned to different values during the training process, and

$$\begin{aligned}\Delta\alpha_{ij}^{(h)} &= -\frac{\partial E}{\partial\alpha_{ij}^{(h)}}\\ &= -\frac{\partial\sum_{k=1}^K(\tilde{y}_k-d_k)^2}{\partial\alpha_{ij}^{(h)}}\\ &= -2\sum_{k=1}^K(\tilde{y}_k-d_k)\frac{\partial\tilde{y}_k}{\partial\alpha_{ij}^{(h)}} \qquad (20)\end{aligned}$$

where $\frac{\partial\tilde{y}_k}{\partial\alpha_{ij}^{(h)}}$ is given by Eq. (21).

IV. Learning Computational Verb Rules by Solving Nonlinear Equations

By representing all computational verbs in a computational verb algorithm by using canonical computational verbs, the learning problem of computational verbs is efficiently transformed into a problem in parameter space. In the parameter space, each computational verb algorithm is defined by a unique parameter vector. Therefore, to learn a computational verb algorithm based on a given set of training sample can be viewed as solving the unknown parameter vector for a set of nonlinear functions. There are two steps to learning computational verb algorithms in this way:

1) Based on the training samples, constructing a set of nonlinear equations with unknown parameter vector as the variables to solve.
2) Solving the set of nonlinear equations numerically.

$$\frac{\partial \tilde{y}_k}{\partial \alpha_{ij}^{(h)}} = \frac{y_i \displaystyle\prod_{g=1,g\neq h}^{p} S(u_{kg}(t),\mathcal{E}_{ig}(t))}{\displaystyle\sum_{i=1}^{n}\prod_{g=1}^{p} S(u_{kg}(t),\mathcal{E}_{ig}(t))}\frac{\partial S(u_{kh}(t),\mathcal{E}_{ih}(t))}{\partial \alpha_{ij}^{(h)}} - \frac{\displaystyle\sum_{i=1}^{n} y_i\prod_{g=1}^{p} S(u_{kg}(t),\mathcal{E}_{ig}(t))}{\left[\displaystyle\sum_{i=1}^{n}\prod_{g=1}^{p} S(u_{kg}(t),\mathcal{E}_{ig}(t))\right]^2}\prod_{g=1,g\neq h}^{p} S(u_{kg}(t),\mathcal{E}_{ig}(t))\frac{\partial S(u_{kh}(t),\mathcal{E}_{ih}(t))}{\partial \alpha_{ij}^{(h)}} \tag{21}$$

where $\frac{\partial S(u_{kh}(t),\mathcal{E}_{ih}(t))}{\partial \alpha_{ij}^{(h)}}$ is given by

$$\begin{aligned}\frac{\partial S(u_{kh}(t),\mathcal{E}_{ih}(t))}{\partial \alpha_{ij}^{(h)}} &= \frac{\partial\left\{1-\sqrt{\frac{1}{T}\int_0^T\left[s_t\left(u_{kh}(t)\right)-s_t\left(\sum_{j=1}^{m}\alpha_{ij}^{(h)}\widetilde{\mathcal{E}}_j(t)\right)\right]^2 dt}\right\}}{\partial \alpha_{ij}^{(h)}} \\ &= \frac{\int_0^T\left[s_t\left(u_{kh}(t)\right)-s_t\left(\sum_{j=1}^{m}\alpha_{ij}^{(h)}\widetilde{\mathcal{E}}_j(t)\right)\right]\dot{s}_t\left(\sum_{j=1}^{m}\alpha_{ij}^{(h)}\widetilde{\mathcal{E}}_j(t)\right)\widetilde{\mathcal{E}}_j(t)dt}{T\sqrt{\frac{1}{T}\int_0^T\left[s_t(u_{kh}(t))-s_t\left(\sum_{j=1}^{m}\alpha_{ij}^{(h)}\widetilde{\mathcal{E}}_j(t)\right)\right]^2 dt}}.\end{aligned} \tag{22}$$

As an illustrative example, let us construct the parameter vector for computational verb algorithm (3) as follow. Let us define the following parameter vector

$$\boldsymbol{\alpha} = (\underbrace{\alpha_{11},\dots,\alpha_{1m}}_{\mathsf{V}_1},\underbrace{\alpha_{21},\dots,\alpha_{2m}}_{\mathsf{V}_2},\dots,\underbrace{\alpha_{n1},\dots,\alpha_{nm}}_{\mathsf{V}_n})^\top. \tag{23}$$

In Eq. (8), let

$$f_k(\boldsymbol{\alpha}) \triangleq \tilde{y}_k - d_k \tag{24}$$

then to find the minimum value of the error function E is equivalent to solve the following set of nonlinear equations

$$\boldsymbol{f}(\boldsymbol{\alpha}) \triangleq \begin{pmatrix} f_1(\boldsymbol{\alpha}) \\ \vdots \\ f_K(\boldsymbol{\alpha}) \end{pmatrix} = \begin{pmatrix} \tilde{y}_1 - d_1 \\ \vdots \\ \tilde{y}_K - d_K \end{pmatrix} = \mathbf{0}. \tag{25}$$

For computational verb algorithm (14), the parameter vector is given by

$$\begin{aligned}\boldsymbol{\alpha} = &(\underbrace{\alpha_{11}^{(1)},\dots,\alpha_{1m}^{(1)}}_{\mathsf{V}_{11}},\underbrace{\alpha_{11}^{(2)},\dots,\alpha_{1m}^{(2)}}_{\mathsf{V}_{12}},\dots,\underbrace{\alpha_{11}^{(p)},\dots,\alpha_{1m}^{(p)}}_{\mathsf{V}_{1p}}, \\ &\underbrace{\alpha_{21}^{(1)},\dots,\alpha_{2m}^{(1)}}_{\mathsf{V}_{21}},\underbrace{\alpha_{21}^{(2)},\dots,\alpha_{2m}^{(2)}}_{\mathsf{V}_{22}},\dots,\underbrace{\alpha_{21}^{(p)},\dots,\alpha_{2m}^{(p)}}_{\mathsf{V}_{2p}}, \\ &\dots,\underbrace{\alpha_{n1}^{(1)},\dots,\alpha_{nm}^{(1)}}_{\mathsf{V}_{n1}},\underbrace{\alpha_{n1}^{(2)},\dots,\alpha_{nm}^{(2)}}_{\mathsf{V}_{n2}},\dots,\underbrace{\alpha_{n1}^{(p)},\dots,\alpha_{nm}^{(p)}}_{\mathsf{V}_{np}}).\end{aligned}$$

The parameter vector for computational verb algorithms (56), (62), (67) and (70) is the same as that for computational verb algorithm (3). The parameter vector for computational verb algorithm (73) is given by

$$\begin{aligned}\boldsymbol{\alpha} = &(\underbrace{\alpha_{11}^{(x)},\dots,\alpha_{1m_x}^{(x)}}_{\mathsf{V}_1^x},\underbrace{\alpha_{21}^{(x)},\dots,\alpha_{2m_x}^{(x)}}_{\mathsf{V}_2^x},\dots,\underbrace{\alpha_{n1}^{(x)},\dots,\alpha_{nm_x}^{(x)}}_{\mathsf{V}_n^x}, \\ &\underbrace{\alpha_{11}^{(y)},\dots,\alpha_{1m_y}^{(y)}}_{\mathsf{V}_1^y},\underbrace{\alpha_{21}^{(y)},\dots,\alpha_{2m_y}^{(y)}}_{\mathsf{V}_2^y},\dots,\underbrace{\alpha_{n1}^{(y)},\dots,\alpha_{nm_y}^{(y)}}_{\mathsf{V}_n^y}).\end{aligned}$$

The parameter vector for computational verb algorithm (83) is given by

$$\begin{aligned}\boldsymbol{\alpha} = &(\underbrace{\alpha_{11}^{(1x)},\dots,\alpha_{1m_x}^{(1x)}}_{\mathsf{V}_{11}^x},\underbrace{\alpha_{11}^{(2x)},\dots,\alpha_{1m_x}^{(2x)}}_{\mathsf{V}_{12}^x},\dots,\underbrace{\alpha_{11}^{(px)},\dots,\alpha_{1m_x}^{(px)}}_{\mathsf{V}_{1p}^x}, \\ &\dots,\underbrace{\alpha_{n1}^{(1x)},\dots,\alpha_{nm_x}^{(1x)}}_{\mathsf{V}_{n1}^x},\underbrace{\alpha_{n1}^{(2x)},\dots,\alpha_{nm_x}^{(2x)}}_{\mathsf{V}_{n2}^x}, \\ &\dots,\underbrace{\alpha_{n1}^{(px)},\dots,\alpha_{nm_x}^{(px)}}_{\mathsf{V}_{np}^x}, \\ &\underbrace{\alpha_{11}^{(y)},\dots,\alpha_{1m_y}^{(y)}}_{\mathsf{V}_1^y},\underbrace{\alpha_{21}^{(y)},\dots,\alpha_{2m_y}^{(y)}}_{\mathsf{V}_2^y},\dots,\underbrace{\alpha_{n1}^{(y)},\dots,\alpha_{nm_y}^{(y)}}_{\mathsf{V}_n^y}).\end{aligned}$$

Here are present some numerical methods of solving $\boldsymbol{f}(\boldsymbol{\alpha}) = \mathbf{0}$.

A. Newton's Method

The most straightforward method of solving Eq. (25) is the Newton's method addressed as follows. In the neighborhood of $\boldsymbol{\alpha}$, $\boldsymbol{f}$ can be expanded in Taylor series

$$\boldsymbol{f}(\boldsymbol{\alpha}+\delta\boldsymbol{\alpha}) = \boldsymbol{f}(\boldsymbol{\alpha}) + \boldsymbol{J}(\boldsymbol{\alpha})\delta\boldsymbol{\alpha} + O(\delta\boldsymbol{\alpha}^2) \tag{26}$$

where $\boldsymbol{J}(\boldsymbol{\alpha})$ is the Jacobian matrix at $\boldsymbol{\alpha}$. In Eq. (26) if we ignore terms of order $\delta\boldsymbol{\alpha}^2$ and higher and let $\boldsymbol{f}(\boldsymbol{\alpha}+\delta\boldsymbol{\alpha}) = \boldsymbol{0}$ then we have the Newton step $\delta\boldsymbol{\alpha} = -\boldsymbol{J}^{-1}(\boldsymbol{\alpha})\boldsymbol{f}(\boldsymbol{\alpha})$. We can find the solution to Eq. (25) by iterating the following equations to convergence.

$$\begin{aligned}\boldsymbol{\alpha}(k+1) &= \boldsymbol{\alpha}(k) - \boldsymbol{J}^{-1}(\boldsymbol{\alpha})\boldsymbol{f}(\boldsymbol{\alpha}), \\ &\quad \text{given an initial condition } \boldsymbol{\alpha}(0). \end{aligned} \tag{27}$$

However, the Newton's method is very sensitive to initial conditions. To overcome this issue, we can construct a globally convergent iterating algorithm as follow. Let

$$\varpi = \frac{1}{2}\boldsymbol{f}(\boldsymbol{\alpha})^\top \boldsymbol{f}(\boldsymbol{\alpha}),$$

then a globally convergent iterating algorithm can be designed when every iterating step decreases ϖ. By doing so, we can guarantee that ϖ keeps decreasing at each iteration. Let $\nabla\varpi$ be gradient (vector of first partial derivatives), since

$$(\nabla\varpi)^\top \delta\boldsymbol{\alpha} = -\boldsymbol{f}(\boldsymbol{\alpha})^\top \boldsymbol{f}(\boldsymbol{\alpha}) \le 0$$

and equality is only satisfied at the solution, the Newton step is a descent direction of ϖ. Therefore, we design the iterating algorithm by searching along the Newton step such that ϖ decreases a big enough amount; namely, to find such a suitable λ in the following equation

$$\boldsymbol{\alpha}(k+1) = \boldsymbol{\alpha}(k) - \lambda \boldsymbol{J}^{-1}(\boldsymbol{\alpha})\boldsymbol{f}(\boldsymbol{\alpha}), \quad k = 0, \ldots \tag{28}$$

that

$$\varpi(\boldsymbol{\alpha}(k)) - \varpi(\boldsymbol{\alpha}(k+1)) > \varepsilon(k), \quad k = 0, \ldots \tag{29}$$

where $\varepsilon(k) > 0$ is a given small value. To achieve this goal, we apply the method presented in [8] as follow. To prevent ϖ from deceasing too slow with respect to the step length, we make the average rate of decreasing ϖ to be related to the initial rate of decreasing $-(\nabla\varpi)^\top \boldsymbol{J}^{-1}(\boldsymbol{\alpha})\boldsymbol{f}(\boldsymbol{\alpha})$ given by

$$\varpi(\boldsymbol{\alpha}(k+1)) \le \varpi(\boldsymbol{\alpha}(k)) + \beta(\nabla\varpi)^\top(\boldsymbol{\alpha}(k+1) - \boldsymbol{\alpha}(k)) \tag{30}$$

where $\beta \in (0,1)$ is a small value of a typical value $\beta = 10^{-4}$. Then a suitable λ can be found by using the following backtracking algorithm.

1) Construct a function

$$g(\lambda,k) = \varpi(\boldsymbol{\alpha}(k) - \lambda \boldsymbol{J}^{-1}(\boldsymbol{\alpha}(k))\boldsymbol{f}(\boldsymbol{\alpha}(k))) \tag{31}$$

and find

$$\begin{aligned}\dot{g}(\lambda,k) &\triangleq \frac{dg(\lambda,k)}{d\lambda} \\ &= -\left(\nabla\varpi(\boldsymbol{\alpha}(k) - \lambda \boldsymbol{J}^{-1}(\boldsymbol{\alpha}(k))\boldsymbol{f}(\boldsymbol{\alpha}(k)))\right)^\top \\ &\quad (\boldsymbol{J}^{-1}(\boldsymbol{\alpha}(k))\boldsymbol{f}(\boldsymbol{\alpha}(k)))). \end{aligned} \tag{32}$$

The first step is to try $\lambda = 1$. If the result is not acceptable then we go to next step.

2) The first backtrack. If $\lambda = 1$ failed, then we begin backtrack. Let us expand $g(\lambda,k)$ into

$$\begin{aligned} g(\lambda,k) &\approx [g(1,k) - g(0,k) - \dot{g}(0,k)]\lambda^2 \\ &\quad + \dot{g}(0,k)\lambda + g(0), \end{aligned} \tag{33}$$

of which the minimum is at

$$\lambda_* = -\frac{\dot{g}(0,k)}{2[g(1,k) - g(0,k) - \dot{g}(0,k)]}. \tag{34}$$

Let $\lambda_2 = 1$, for a small enough β we choose $\lambda_1 = \min(0.5, \max(\lambda_*, 0.1))$. If we need further backtrack, then we go to next step.

3) Repeat the following steps until no further backtrack needed.

   a) Solve a and b from the following equations.

$$\begin{aligned}\begin{pmatrix} a \\ b \end{pmatrix} &= \frac{1}{\lambda_1 - \lambda_2}\begin{pmatrix} 1/\lambda_1^2 & -1/\lambda_2^2 \\ -\lambda_2/\lambda_1^2 & \lambda_1/\lambda_2^2 \end{pmatrix} \\ &\quad \begin{pmatrix} g(\lambda_1) - \dot{g}(0)\lambda_1 - g(0) \\ g(\lambda_2) - \dot{g}(0)\lambda_2 - g(0) \end{pmatrix}. \end{aligned} \tag{35}$$

   Then the minimum of function $g(\lambda) = a\lambda^3 + b\lambda^2 + \dot{g}(0)\lambda + g(0)$ is at

$$\lambda_* = \frac{-b + \sqrt{b^2 - 3a\dot{g}(0)}}{3a}; \tag{36}$$

   b) $\lambda_2 = \lambda_1$;

   c) $\lambda_1 = \min(0.5\lambda_1, \max(\lambda_*, 0.1\lambda_1))$.

B. Broyden's Method

One drawback of Newton's method is that the Jacobian matrix is needed. In cases when the Jacobian matrix is not analytically available or too expensive to calculate, we use Broyden's method[8] to find approximate Jacobian. Let $\boldsymbol{B}(k)$ be the approximate Jacobian at the kth iteration, then we have

$$\boldsymbol{B}(k)\delta\boldsymbol{\alpha}(k) = -\boldsymbol{f}(\boldsymbol{\alpha}(k)) \tag{37}$$

where $\delta\boldsymbol{\alpha}(k) = \boldsymbol{\alpha}(k+1) - \boldsymbol{\alpha}(k)$. Then the quasi-Newton condition is that $\boldsymbol{B}(k+1)$ satisfies

$$\boldsymbol{B}(k+1)\delta\boldsymbol{\alpha}(k) = \delta\boldsymbol{f}(\boldsymbol{\alpha}(k)) \triangleq \boldsymbol{f}(\boldsymbol{\alpha}(k+1)) - \boldsymbol{f}(\boldsymbol{\alpha}(k)). \tag{38}$$

In Broyden's method we use the following formula to determine $\boldsymbol{B}(k+1)$

$$\boldsymbol{B}(k+1) = \boldsymbol{B}(k) + \frac{[\delta\boldsymbol{f}(\boldsymbol{\alpha}(k)) - \boldsymbol{B}(k)\cdot\delta\boldsymbol{\alpha}(k)] \times \delta\boldsymbol{\alpha}(k)}{\delta\boldsymbol{\alpha}(k)\cdot\delta\boldsymbol{\alpha}(k)}. \tag{39}$$

The initial condition $\boldsymbol{B}(0)$ is usually chosen as the identity matrix.

V. Learning as Finding Minimum of Error Function

To learn computational verbs for each antecedent of a computational verb algorithm is to find a minimum of an error function, e.g., Eq. (8). Here two more methods will be used to find the minimum of the error function. As an illustrating example, we study a simple computational verb algorithm, of which each antecedent consists of a single computational verb and each consequent is a real number.

Let us assume that there are $m = 3$ canonical computational verbs, of which the evolving functions are given by

$$\begin{aligned} \widetilde{\mathcal{E}}_1(t) &= 1 - e^{-t}, \\ \widetilde{\mathcal{E}}_2(t) &= 0.5, \\ \widetilde{\mathcal{E}}_3(t) &= e^{-t}, t \in [0, 10]. \end{aligned} \tag{40}$$

The evolving functions of the three canonical computational verbs are shown in Fig. 1.

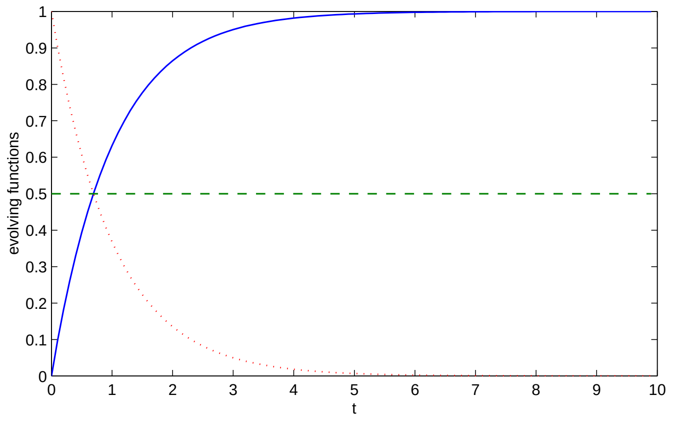

Fig. 1. The three canonical computational verbs in Eq. (40). The evolving functions $\widetilde{\mathcal{E}}_1$(solid), $\widetilde{\mathcal{E}}_2$(dashed), and $\widetilde{\mathcal{E}}_3$(dotted) are shown.

Let's study the following three computational verb rules

$$\begin{aligned} &\text{IF } x(t) \text{ increases, then } y \text{ is } 0.9; \\ &\text{IF } x(t) \text{ stays, then } y \text{ is } 0.5; \\ &\text{IF } x(t) \text{ decreases, then } y \text{ is } 0.1. \end{aligned} \tag{41}$$

The following six samples of computational verbs are chosen to train the computational verb rules.

$$\begin{aligned} \mathcal{E}_{\text{increase } 1}(t) &= 1 - e^{-1.1t}, y_1 = 0.9, \\ \mathcal{E}_{\text{increase } 2}(t) &= 1 - e^{-0.9t}, y_2 = 0.9, \\ \mathcal{E}_{\text{decrease } 1}(t) &= e^{-1.1t}, y_3 = 0.1, \\ \mathcal{E}_{\text{decrease } 2}(t) &= e^{-0.9t}, y_4 = 0.1, \\ \mathcal{E}_{\text{stay } 1}(t) &= 0.6, y_5 = 0.5, \\ \mathcal{E}_{\text{stay } 2}(t) &= 0.4, y_6 = 0.5, t \in [0, 10]. \end{aligned} \tag{42}$$

The computational verb similarity is calculated at samples of the continuous evolving functions as

$$\begin{aligned} S(\mathcal{E}_1, \mathcal{E}_2) &= \frac{1}{100} \sum_{i=0}^{99} \left(\frac{\min(s_t(\mathcal{E}_1(i\delta)), s_t(\mathcal{E}_2(i\delta)))}{\max(s_t(\mathcal{E}_1(i\delta)), s_t(\mathcal{E}_2(i\delta)))} \right)^{100}, \\ &\delta = 0.1. \end{aligned} \tag{43}$$

Let the parameter vector be

$$\boldsymbol{\alpha} = (\alpha_{11}\ \alpha_{12}\ \alpha_{13}\ \alpha_{21}\ \alpha_{22}\ \alpha_{23}\ \alpha_{31}\ \alpha_{32}\ \alpha_{33})^\top. \tag{44}$$

Then the three computational verbs in the antecedents are given by

$$\begin{aligned} \mathcal{E}_{\text{increase}}(t) &= \alpha_{11}(1 - e^{-t}) + 0.5\alpha_{12} + \alpha_{13}e^{-t}, \\ \mathcal{E}_{\text{stay}}(t) &= \alpha_{21}(1 - e^{-t}) + 0.5\alpha_{22} + \alpha_{23}e^{-t}, \\ \mathcal{E}_{\text{decrease}}(t) &= \alpha_{31}(1 - e^{-t}) + 0.5\alpha_{32} + \alpha_{33}e^{-t}, t \in [0, 10]. \end{aligned} \tag{45}$$

A. Downhill Simplex Method

Downhill simplex method is an easy way to find the minimum of a function as presented in [8]. Since this method requires only function evaluations but not derivatives, it eases the implementation at the price of moderate computational complexity. In the parameter space of a verb rule set, if the parameter vectors are of n-dimension, then a *simplex* is a geometrical structure consisting of $(n + 1)$ vertices. The downhill simplex method begins with an initial simplex, which consists of an initial starting parameter vector $\boldsymbol{\alpha}_0$ and the other n parameter vectors are generated as follows

$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_0 + \delta_i \boldsymbol{e}_i \tag{46}$$

where $\boldsymbol{e}_i$'s are n unit vectors of the parameter space, and δ_i's are constants related to the guessed range than encloses a minimum. The basic idea of downhill simplex method is to reallocate the vertex of high error to reduce the entire level of errors at all simplex vertices. The configurations of simplex change at each iteration are shown in Fig. 2. Figure 2(a) shows the simplex at the beginning of one iteration when some vertices are of high values and the others are of low values. At the end of the iteration, the resulting simplex can be any one of those shown in Figs. 2(b) to (e). In Figs. 2(b) to (e), the original simplex is shown in dashed lines for comparison.

To learn parameters in computational verb algorithm (41), the initial starting parameter vector is given by

$$\boldsymbol{\alpha}_0 = (1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1)^\top. \tag{47}$$

And other 9 starting parameter vectors are

$$\begin{aligned} \boldsymbol{\alpha}_1 &= (2\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1)^\top, \\ \boldsymbol{\alpha}_2 &= (1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1)^\top, \\ \boldsymbol{\alpha}_3 &= (1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1)^\top, \\ \boldsymbol{\alpha}_4 &= (1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1)^\top, \\ \boldsymbol{\alpha}_5 &= (1\ 0\ 0\ 0\ 2\ 0\ 0\ 0\ 1)^\top, \\ \boldsymbol{\alpha}_6 &= (1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1)^\top, \\ \boldsymbol{\alpha}_7 &= (1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1)^\top, \\ \boldsymbol{\alpha}_8 &= (1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1)^\top, \\ \boldsymbol{\alpha}_9 &= (1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 2)^\top. \end{aligned} \tag{48}$$

The learning result is

$$\begin{aligned} \boldsymbol{\alpha}_* = &(0.999829\ -0.000053\ 0.010922 \\ &0.065448\ 1.048120\ 0.638260 \\ &-0.005613\ 0.011713\ 1.228373)^\top. \end{aligned} \tag{49}$$

The error is $e = 0.000318$ for this parameter vector. The evolving functions of the three learnt computational verbs

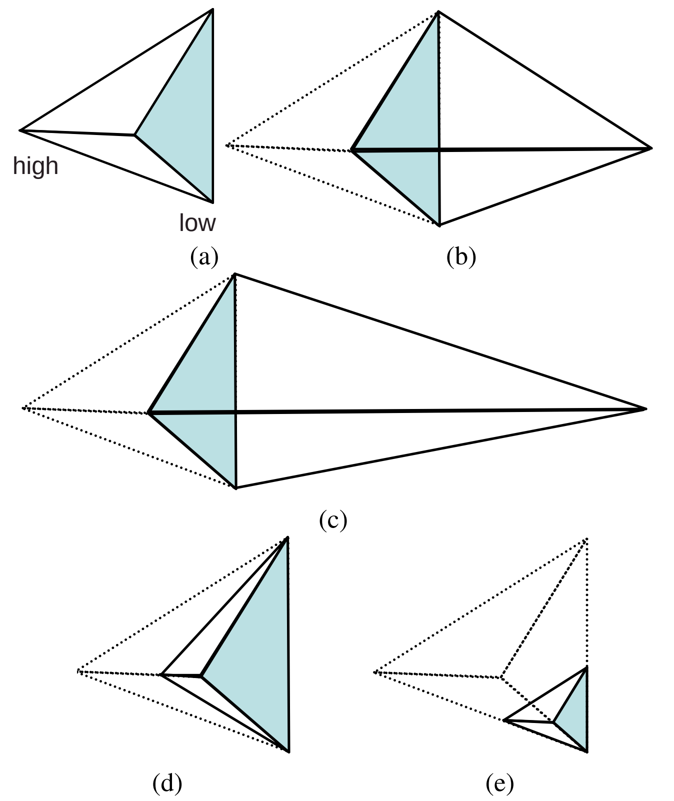

Fig. 2. All possible transformations of a simplex in one iteration of downhill simplex method. (a) The original simplex at the beginning of one iteration. (b) A reflection towards opposite direction away from the high point. (c) Same as (b) with expansion. (d) A contraction along one dimension leaving the high point. (e) A contraction along all dimensions approaching the low point.

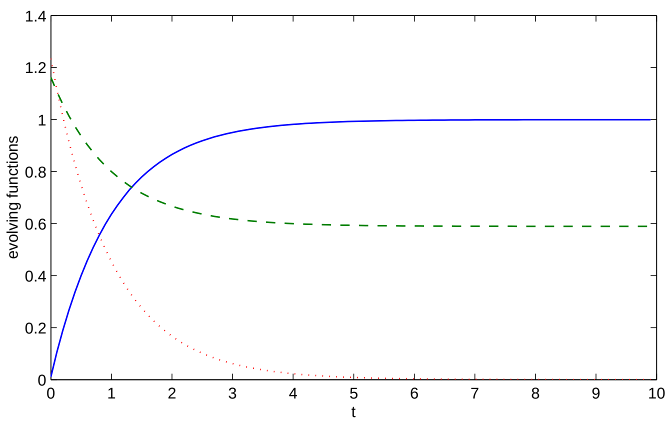

Fig. 3. The three learnt computational verbs in the antecedents in the computational verb algorithm (41) by using downhill simplex method. The evolving functions for increase(solid), stay(dashed), and decrease(dotted) are shown.

in the antecedents of computational verb algorithm (41) are shown in Fig. 3. Observe that the worse learning result is in that of stay. The first $1/3$ of the evolving function of stay is in fact a part of decrease. However, the rest of stay is very near the ideal situation.

B. Direction Set (Powell's) Method

As addressed in [8], when we use Powell's method to find a minimum of a function, we choose a set of directions and move along the first direction to its minimum, then from there along the second direction to its minimum and repeat this procedure for the rest directions. This procedure is repeated as many times as necessary until the minimum becomes unchanged. The key to a good performance of Powell's method is to choose a set of *good* directions. For a given parameter vector $\boldsymbol{\alpha}_*$, in its neighborhood any function f can be approximated by its Taylor series as

$$\begin{aligned} f(\boldsymbol{\alpha}) &= f(\boldsymbol{\alpha}_*) + \sum_i \frac{\partial f}{\partial \alpha_i}\alpha_i + \frac{1}{2}\sum_{i,j}\frac{\partial^2 f}{\partial\alpha_i\partial\alpha_j}\alpha_i\alpha_j + \dots \\ &\approx c - \boldsymbol{b}\cdot\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}\cdot\boldsymbol{H}\cdot\boldsymbol{\alpha} \end{aligned} \tag{50}$$

where

$$c \triangleq f(\boldsymbol{\alpha}_*), \quad \boldsymbol{b} \triangleq -\nabla f|_{\boldsymbol{\alpha}_*}, \quad [\boldsymbol{H}]_{ij} \triangleq \left.\frac{\partial^2 f}{\partial\alpha_i\partial\alpha_j}\right|_{\boldsymbol{\alpha}_*}. \tag{51}$$

The matrix $\boldsymbol{H}$ is known as the *Hessian matrix* of the function f at $\boldsymbol{\alpha}_*$. It follows from Eq. (50) that the gradient of f can be calculated as

$$\nabla f|_{\boldsymbol{\alpha}} = \nabla\left(c - \boldsymbol{b}\cdot\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}\cdot\boldsymbol{H}\cdot\boldsymbol{\alpha}\right) = \boldsymbol{H}\cdot\boldsymbol{\alpha} - \boldsymbol{b} \tag{52}$$

which implies that the function f will be at an extremum at the solution of $\boldsymbol{H}\cdot\boldsymbol{\alpha} = \boldsymbol{b}$. It follows from Eq. (52) that if we move along some direction, the change of gradient is given by

$$\delta(\nabla f) = \boldsymbol{H}\cdot(\delta\boldsymbol{\alpha}). \tag{53}$$

Assume that we have moved along a direction $\boldsymbol{u}$ to a minimum of f and now to move along the second direction $\boldsymbol{v}$. To prevent moving along $\boldsymbol{v}$ from spoiling the minimization along $\boldsymbol{u}$, the change in the gradient should be perpendicular to $\boldsymbol{u}$; namely, it follows from Eq. (53) that

$$0 = \boldsymbol{u}\cdot\delta(\nabla f) = \boldsymbol{u}\cdot\boldsymbol{H}\cdot\boldsymbol{v}. \tag{54}$$

Two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ that satisfy Eq. (54) are *conjugate*. Give a set of vectors, if its elements are conjugate pairwise, then this set is a *conjugate set*. The importance of the concept of conjugate set to minimization of a function f is that if we do successive line minimizations of f along a conjugate set of directions, then the minimization along each direction is decoupled to the others in the conjugate set. Therefore, the key is to find such a conjugate set of directions. Consult [8] for a detailed description of how to construct this kind of conjugate set of directions in Powell's method.

By using Powell's method, after 14 iterations, we arrived at the following solution

$$\begin{aligned} \boldsymbol{\alpha}_* = &(0.987592\ 0.019557\ 0.002694 \\ &0.093044\ 1.018511\ \ -2.188152 \\ &-0.002231\ 0.005155\ 1.007018)^\top. \end{aligned} \tag{55}$$

The minimum value of the error function is 0.000164. The initial parameter vector is the same as that in Eq. (47). The evolving functions of the three learnt computational verbs in the antecedents of computational verb algorithm (41) are shown in Fig. 4. Also, observe that the biggest error is contributed by stay.

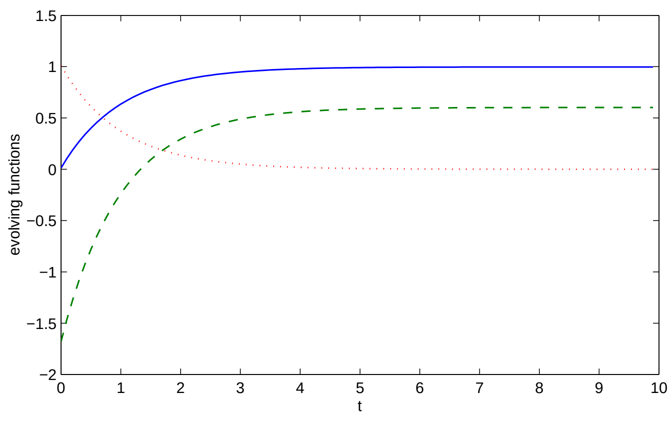

Fig. 4. The three learnt computational verbs in the antecedents in the computational verb algorithm (41) by using Powell's method. The evolving functions for increase(solid), stay(dashed), and decrease(dotted) are shown.

VI. Learning Computational Verb Rules with Static Antecedents

Here we study the computational verb rules of the form in Eq. (2).

A. Single-input Computational Verb Algorithms

Let's assume the following computational verb algorithm of n computational verb rules.

$$\begin{aligned} &\text{IF } x \text{ is } x_1, \text{ THEN } y(t)\ \mathsf{V}_1;\\ &\vdots\\ &\text{IF } x \text{ is } x_n, \text{ THEN } y(t)\ \mathsf{V}_n, \end{aligned} \qquad (56)$$

where $y(t) \in \mathbb{R}, t \in [0,T]$ is a measured waveform known as *output computational verb* and $x_i \in \mathbb{R}$ is a *real input*. $\mathsf{V}_i, i = 1,\dots,n$, are n computational verbs, of which the evolving functions are $\mathcal{E}_i(t) \in \mathbb{R}, t \in [0,T], i = 1,\dots,n$. Observe that each computational verb rule in algorithm (56) consists of an antecedent of a real number and a consequent of one computational verb.

Assume that we observe an input $x = a$, then the output of the entire algorithm is calculated by

$$y(t) = \frac{\sum_{i=1}^{n} s_t\left(\frac{1}{|x_i - a|}\right)\mathcal{E}_i(t)}{\sum_{i=1}^{n} s_t\left(\frac{1}{|x_1 - a|}\right)}. \qquad (57)$$

Assume that all computational verbs in the consequents satisfy Eq. (7), and given K sets of training samples $\{(u_k, d_k(t))\}_{k=1}^{K}$, we define an error function as

$$E = \sum_{k=1}^{K}\int_0^T [\tilde{y}_k(t) - d_k(t)]^2 dt \qquad (58)$$

where

$$\tilde{y}_k(t) = \frac{\sum_{i=1}^{n} s_t\left(\frac{1}{|x_i - u_k|}\right)\mathcal{E}_i(t)}{\sum_{i=1}^{n} s_t\left(\frac{1}{|x_1 - u_k|}\right)}. \qquad (59)$$

Let us assume $\mathsf{V}_i, i = 1,\dots,n$, satisfy Eq. (7), then the parameters for computational verb algorithm (56) are $\alpha_{ij}, i = 1,\dots,n; j = 1,\dots,m$. We choose the learning rule (10) for α_{ij}, $\Delta\alpha_{ij}$ can be calculated as

$$\begin{aligned} \Delta\alpha_{ij} &= -\frac{\partial E}{\partial \alpha_{ij}}\\ &= -\frac{\partial \sum_{k=1}^{K}\int_0^T [\tilde{y}_k(t) - d_k(t)]^2 dt}{\partial \alpha_{ij}}\\ &= -2\sum_{k=1}^{K}\int_0^T [\tilde{y}_k(t) - d_k(t)]\frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}} dt \end{aligned} \qquad (60)$$

where

$$\begin{aligned} \frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}} &= \frac{\partial \dfrac{\sum_{i=1}^{n} s_t\left(\frac{1}{|x_i - u_k|}\right)\mathcal{E}_i(t)}{\sum_{i=1}^{n} s_t\left(\frac{1}{|x_i - u_k|}\right)}}{\partial \alpha_{ij}}\\ &= \frac{s_t\left(\dfrac{1}{|x_i - u_k|}\right)\dfrac{\partial \mathcal{E}_i(t)}{\partial \alpha_{ij}}}{\sum_{i=1}^{n} s_t\left(\dfrac{1}{|x_i - u_k|}\right)}\\ &= \frac{s_t\left(\dfrac{1}{|x_i - u_k|}\right)\widetilde{\mathcal{E}}_j(t)}{\sum_{i=1}^{n} s_t\left(\dfrac{1}{|x_i - u_k|}\right)}. \end{aligned} \qquad (61)$$

B. Multi-input Computational Verb Algorithms

Let's assume the following computational verb algorithm of n computational verb rules.

$$\begin{aligned} &\text{IF } x_1 \text{ is } x_{11} \text{ AND } x_2 \text{ is } x_{12} \dots \text{AND } x_p \text{ is } x_{1p},\\ &\quad \text{THEN } y_1(t)\ \mathsf{V}_1;\\ &\vdots\\ &\text{IF } x_1 \text{ is } x_{n1} \text{ AND } x_2 \text{ is } x_{n2} \dots \text{AND } x_p \text{ is } x_{np},\\ &\quad \text{THEN } y_n(t)\ \mathsf{V}_n. \end{aligned} \qquad (62)$$

where $x_i \in \mathbb{R}, i = 1,\dots,p$ are p inputs. $\mathsf{V}_i, i = 1,\dots,n$, are n computational verbs of which the evolving functions are $\mathcal{E}_i(t) \in \mathbb{R}, t \in [0,T], i = 1,\dots,n$. Observe that the antecedent of each computational verb rule consists of p real numbers while the consequent is a computational verb.

Assume that we observe a p-vector of input $(a_1,\dots,a_p)$, then the output of this multi-input computational algorithm is given by

$$y(t) = \frac{\sum_{i=1}^{n}\mathcal{E}_i(t)\prod_{j=1}^{p} s_t\left(\frac{1}{x_{ij} - a_j}\right)}{\sum_{i=1}^{n}\prod_{j=1}^{p} s_t\left(\frac{1}{x_{ij} - a_j}\right)}. \qquad (63)$$

Assume K sets of training samples $\{(u_{k1},\dots,u_{kp},d_k(t))\}_{k=1}^{K}$, then we have

$$\tilde{y}_k(t)=\frac{\sum_{i=1}^{n}\mathcal{E}_i(t)\prod_{j=1}^{p}s_t\left(\frac{1}{x_{ij}-u_{kj}}\right)}{\sum_{i=1}^{n}\prod_{j=1}^{p}s_t\left(\frac{1}{x_{ij}-u_{kj}}\right)}. \quad (64)$$

Similarly, let us assume $\mathsf{V}_i, i=1,\dots,n$ satisfy Eq. (7), then the parameters for computational verb algorithm (62) are $\alpha_{ih}, i=1,\dots,n; h=1,\dots,m$. To learn α_{ih} based on the learning rule (10) and the error function (58), $\Delta\alpha_{ih}$ can be calculated as

$$\begin{aligned}\Delta\alpha_{ih} &= -\frac{\partial E}{\partial\alpha_{ih}}\\ &= -\frac{\partial\sum_{k=1}^{K}\int_0^T[\tilde{y}_k(t)-d_k(t)]^2dt}{\partial\alpha_{ih}}\\ &= -2\sum_{k=1}^{K}\int_0^T[\tilde{y}_k(t)-d_k(t)]\frac{\partial\tilde{y}_k(t)}{\partial\alpha_{ih}}dt \quad (65)\end{aligned}$$

where

$$\begin{aligned}\frac{\partial\tilde{y}_k(t)}{\partial\alpha_{ih}} &= \frac{\frac{\mathcal{E}_i(t)}{\partial\alpha_{ih}}\prod_{j=1}^{p}s_t\left(\frac{1}{x_{ij}-u_{kj}}\right)}{\sum_{i=1}^{n}\prod_{j=1}^{p}s_t\left(\frac{1}{x_{ij}-u_{kj}}\right)}\\ &= \frac{\widetilde{\mathcal{E}}_h(t)\prod_{j=1}^{p}s_t\left(\frac{1}{x_{ij}-u_{kj}}\right)}{\sum_{i=1}^{n}\prod_{j=1}^{p}s_t\left(\frac{1}{x_{ij}-u_{kj}}\right)}. \quad (66)\end{aligned}$$

C. Fuzzification

The antecedents of rules in computational verb algorithms (56) and (62) are *attribute values*, which are represented as real numbers. Besides real numbers, fuzzy membership functions and probability density functions can also function as attribute values. Therefore, all real numbers in computational verb algorithms (56) and (62) can be replaced by fuzzy membership functions as follows.

1) Computational Verb Algorithm (56): Computational verb algorithm (56) can be recast into

IF x is X_1, THEN $y(t)$ V_1;

$\vdots$

IF x is X_n, THEN $y(t)$ V_n (67)

where "is's" are the default computational verbs used in conventional logic systems. $X_i, i=1,\dots,n$, are fuzzy sets of which the membership functions are $\mu_{X_i}(x), i=1,\dots,n$. Equation (59) is recast into

$$\tilde{y}_k(t)=\frac{\sum_{i=1}^{n}\mu_{X_i}(u_k)\mathcal{E}_i(t)}{\sum_{i=1}^{n}\mu_{X_i}(u_k)}. \quad (68)$$

Then the learning algorithm in (60) and (61) becomes

$$\Delta\alpha_{ij} = -2\sum_{k=1}^{K}\int_0^T[\tilde{y}_k(t)-d_k(t)]\frac{\mu_{X_i}(u_k)\widetilde{\mathcal{E}}_j(t)}{\sum_{i=1}^{n}\mu_{X_i}(u_k)}dt. \quad (69)$$

2) Computational Verb Algorithm (62):

IF x_1 is X_{11} AND x_2 is X_{12} …AND x_p is X_{1p}, THEN $y_1(t)$ V_1;

$\vdots$

IF x_1 is X_{n1} AND x_2 is X_{n2} …AND x_p is X_{np}, THEN $y_n(t)$ V_n (70)

where $X_{ij}, i=1,\dots,n; j=1,\dots,p$, are fuzzy sets, of which the membership functions are $\mu_{X_{ij}}(x_i), i=1,\dots,n; j=1,\dots,p$. Equation (64) is recast into

$$\tilde{y}_k(t)=\frac{\sum_{i=1}^{n}\mathcal{E}_i(t)\bigwedge_{j=1}^{p}\mu_{X_{ij}}(u_{kj})}{\sum_{i=1}^{n}\bigwedge_{j=1}^{p}\mu_{X_{ij}}(u_{kj})}. \quad (71)$$

Then the learning algorithm in Eqs. (65) and (66) becomes

$$\Delta\alpha_{ih} = -2\sum_{k=1}^{K}\int_0^T[\tilde{y}_k(t)-d_k(t)]\frac{\widetilde{\mathcal{E}}_h(t)\bigwedge_{j=1}^{p}\mu_{X_{ij}}(u_{kj})}{\sum_{i=1}^{n}\bigwedge_{j=1}^{p}\mu_{X_{ij}}(u_{kj})}dt. \quad (72)$$

VII. Computational Verb Rules with Computational Verb Antecedents and Consequents

So far, we only study the learning algorithm for computational verb rules, of which either antecedents or consequents are attribute values; namely, real numbers or fuzzy membership functions. Here we will construct the learning algorithms for computational verb rules, of which all antecedents and consequents are in the form of computational verbs.

A. Single-input Computational Verb Algorithms

Let's assume the following computational verb algorithm of n computational verb rules.

IF $x(t)$ V_1^x, THEN $y(t)$ V_1^y;

$\vdots$

IF $x(t)$ V_n^x, THEN $y(t)$ V_n^y (73)

where $x(t) \in \mathbb{R}, t \in [0,T]$ is a measured waveform known as *input* and $y(t) \in \mathbb{R}, t \in [0,T]$ is an *output*. In the antecedents, $\mathsf{V}_i^x, i = 1,\ldots,n$, are n computational verbs, of which the evolving functions are $\mathcal{E}_i^x(t) \in \mathbb{R}, t \in [0,T], i = 1,\ldots,n$. In the consequents, $\mathsf{V}_i^y, i = 1,\ldots,n$, are n computational verbs, of which the evolving functions are $\mathcal{E}_i^y(t) \in \mathbb{R}, t \in [0,T], i = 1,\ldots,n$.

Given an observation $x(t)$, the output of this verb algorithm is a computational verb V_y, of which the evolving function $\mathcal{E}_y(t)$ is calculated as

$$\mathcal{E}_y(t) = \frac{\sum_{i=1}^{n} S(x(t),\mathcal{E}_i^x(t))\mathcal{E}_i^y(t)}{\sum_{i=1}^{n} S(x(t),\mathcal{E}_i^x(t))}. \tag{74}$$

We assume that there are m_x and m_y canonical computational verbs to construct V_i^x and $\mathsf{V}_i^y, i = 1,\ldots,n$, respectively. Then the computational verbs in computational verb algorithm (73) can be expressed as

$$\mathcal{E}_i^x(t) = \sum_{j=1}^{m_x} \alpha_{ij}^x \widetilde{\mathcal{E}}_j^x(t), \quad \mathcal{E}_i^y(t) = \sum_{j=1}^{m_y} \alpha_{ij}^y \widetilde{\mathcal{E}}_j^y(t). \tag{75}$$

Given K sets of training samples $\{(u_k(t), d_k(t))\}_{k=1}^K$, we define an error function as

$$E = \sum_{k=1}^{K} \int_0^T [\tilde{y}_k(t) - d_k(t)]^2 dt \tag{76}$$

where

$$\tilde{y}_k(t) = \frac{\sum_{i=1}^{n} S(u_k(t),\mathcal{E}_i^x(t))\mathcal{E}_i^y(t)}{\sum_{i=1}^{n} S(u_k(t),\mathcal{E}_i^x(t))}. \tag{77}$$

The learning rules are given by

$$\begin{aligned} \alpha_{ij}^x(l+1) &= \alpha_{ij}^x(l) + \gamma_{ij}^x(l)\Delta\alpha_{ij}^x(l), \\ \alpha_{ij}^y(l+1) &= \alpha_{ij}^y(l) + \gamma_{ij}^y(l)\Delta\alpha_{ij}^y(l), \quad l = 1,\ldots \end{aligned} \tag{78}$$

where $\gamma_{ij}^x(l) \in \mathbb{R}^+$ and $\gamma_{ij}^y(l) \in \mathbb{R}^+$ are learning rates, which can be tuned to different values during the training process for each parameter, and

$$\begin{aligned} \Delta\alpha_{ij}^x &= -\frac{\partial \sum_{k=1}^{K} \int_0^T [\tilde{y}_k(t) - d_k(t)]^2 dt}{\partial \alpha_{ij}^x} \\ &= -2\sum_{k=1}^{K} \int_0^T [\tilde{y}_k(t) - d_k(t)] \frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}^x} dt \end{aligned} \tag{79}$$

where $\frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}^x}$ is given by Eq. (80) for one of many possible computational verb similarities[36].

$$\begin{aligned} \Delta\alpha_{ij}^y &= -\frac{\partial \sum_{k=1}^{K} \int_0^T [\tilde{y}_k(t) - d_k(t)]^2 dt}{\partial \alpha_{ij}^y} \\ &= -2\sum_{k=1}^{K} \int_0^T [\tilde{y}_k(t) - d_k(t)] \frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}^y} dt \end{aligned} \tag{81}$$

where $\frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}^y}$ is given by

$$\begin{aligned} \frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}^y} &= \frac{\partial \frac{\sum_{i=1}^n S(u_k(t),\mathcal{E}_i^x(t))\mathcal{E}_i^y(t)}{\sum_{i=1}^n S(u_k(t),\mathcal{E}_i^x(t))}}{\partial \alpha_{ij}^y} \\ &= \frac{S(u_k(t),\mathcal{E}_i^x(t))}{\sum_{i=1}^{n} S(u_k(t),\mathcal{E}_i^x(t))} \frac{\partial \mathcal{E}_i^y(t)}{\partial \alpha_{ij}^y} \\ &= \frac{S(u_k(t),\mathcal{E}_i^x(t))}{\sum_{i=1}^{n} S(u_k(t),\mathcal{E}_i^x(t))} \widetilde{\mathcal{E}}_j^y(t). \end{aligned} \tag{82}$$

B. Multi-input Computational Verb Algorithms

Let's assume the following computational verb algorithm of n computational verb rules.

IF $x_1(t)$ V_{11}^x AND $x_2(t)$ V_{12}^x …AND $x_p(t)$ V_{1p}^x,
THEN $y(t)$ V_1^y;

$\vdots$

IF $x_1(t)$ V_{n1}^x AND $x_2(t)$ V_{n2}^x …AND $x_p(t)$ V_{np}^x,
THEN $y(t)$ V_n^y (83)

where $x_i(t) \in \mathbb{R}, t \in [0,T], i = 1,\ldots,p$ are p inputs. In the antecedents, $\mathsf{V}_{ij}^x, i = 1,\ldots,n; j = 1,\ldots,p$, are np computational verbs, of which the evolving functions are $\mathcal{E}_{ij}^x(t) \in \mathbb{R}, t \in [0,T], i = 1,\ldots,n; j = 1,\ldots,p$. In the consequents, $\mathsf{V}_i^y, i = 1,\ldots,n$ are n computational verbs, of which the evolving functions are $\mathcal{E}_i^y(t) \in \mathbb{R}, t \in [0,T], i = 1,\ldots,n$.

The output of this multi-input computational algorithm is a computational verb V_y, of which the evolving function, $\mathcal{E}_y(t)$, is given by

$$\mathcal{E}_y(t) = \frac{\sum_{i=1}^{n} \mathcal{E}_i^y(t) \prod_{j=1}^{p} S(x_j(t),\mathcal{E}_{ij}^x(t))}{\sum_{i=1}^{n} \prod_{j=1}^{p} S(x_j(t),\mathcal{E}_{ij}^x(t))}. \tag{84}$$

We assume that there are m_x canonical computational verbs $\widetilde{\mathcal{E}}_j^x(t), j = 1,\ldots,m_x$, which are used to construct all computational verbs in antecedents as

$$\mathcal{E}_{ih}^x(t) = \sum_{j=1}^{m_x} \alpha_{ij}^{(hx)} \widetilde{\mathcal{E}}_j^x(t), \quad h = 1,\ldots,p, \tag{85}$$

where $\alpha_{ij}^{(hx)} \in \mathbb{R}, i = 1,\ldots,n; j = 1,\ldots,m_x$, function as adverbs. We also assume that there are m_y canonical

$$\begin{aligned}\frac{\partial \tilde{y}_k(t)}{\partial \alpha^x_{ij}} &= \frac{\partial \dfrac{\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))\mathcal{E}^y_i(t)}{\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))}}{\partial \alpha^x_{ij}} \\ &= \left[\frac{\mathcal{E}^y_i(t)}{\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))} - \frac{\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))\mathcal{E}^y_i(t)}{\left[\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))\right]^2}\right]\frac{\partial S(u_k(t),\mathcal{E}^x_i(t))}{\partial \alpha^x_{ij}} \\ &= \left[\frac{\mathcal{E}^y_i(t)}{\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))} - \frac{\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))\mathcal{E}^y_i(t)}{\left[\sum_{i=1}^n S(u_k(t),\mathcal{E}^x_i(t))\right]^2}\right] \\ &\quad\times \frac{\int_0^T \left[s_t(u_k(t)) - s_t\left(\sum_{j=1}^{m_x}\alpha^x_{ij}\widetilde{\mathcal{E}}^x_j(t)\right)\right]\dot{s}_t\left(\sum_{j=1}^{m_x}\alpha^x_{ij}\widetilde{\mathcal{E}}^x_j(t)\right)\widetilde{\mathcal{E}}^x_j(t)dt}{T\sqrt{\frac{1}{T}\int_0^T\left[s_t(u_k(t)) - s_t\left(\sum_{j=1}^{m_x}\alpha^x_{ij}\widetilde{\mathcal{E}}^x_j(t)\right)\right]^2 dt}}. \end{aligned} \quad (80)$$

computational verbs $\widetilde{\mathcal{E}}^y_i(t), i=1,\dots,m_y$, which are used to construct all computational verbs in consequents as

$$\mathcal{E}^y_i(t) = \sum_{j=1}^{m_y}\alpha^y_{ij}\widetilde{\mathcal{E}}^y_j(t), \quad i=1,\dots,n. \quad (86)$$

Given K sets of training samples $\{(u_{k1}(t),\dots,u_{kp}(t),d_k(t)\}_{k=1}^K$, we define an error function as

$$E = \sum_{k=1}^K\int_0^T[\tilde{y}_k(t)-d_k(t)]^2dt \quad (87)$$

where

$$\tilde{y}_k(t) = \frac{\sum_{i=1}^n \mathcal{E}^y_i(t)\prod_{h=1}^p S(u_{kh}(t),\mathcal{E}^x_{ih}(t))}{\sum_{i=1}^n\prod_{h=1}^p S(u_{kh}(t),\mathcal{E}^x_{ih}(t))}. \quad (88)$$

The learning rules are given by

$$\begin{aligned}\alpha^{(hx)}_{ij}(l+1) &= \alpha^{(hx)}_{ij}(l) + \gamma^{(hx)}_{ij}\Delta\alpha^{(hx)}_{ij}(l), \\ \alpha^y_{ij}(l+1) &= \alpha^y_{ij}(l)+\gamma^y_{ij}\Delta\alpha^y_{ij}(l), \quad l=1,\dots, \end{aligned} \quad (89)$$

where $\gamma^{(hx)}_{ij}\in\mathbb{R}^+$ and $\gamma^y_{ij}\in\mathbb{R}^+$ are learning rates, which can be tuned to different values during the training process for each parameter, and

$$\begin{aligned}\Delta\alpha^{(hx)}_{ij} &= -\frac{\partial\sum_{k=1}^K\int_0^T[\tilde{y}_k(t)-d_k(t)]^2dt}{\partial\alpha^{(hx)}_{ij}} \\ &= -2\sum_{k=1}^K\int_0^T[\tilde{y}_k(t)-d_k(t)]\frac{\partial\tilde{y}_k(t)}{\partial\alpha^{(hx)}_{ij}}dt \end{aligned} \quad (90)$$

where $\frac{\partial\tilde{y}_k(t)}{\partial\alpha^{(hx)}_{ij}}$ is given by Eq. (91).

$$\Delta\alpha^y_{ij} = -2\sum_{k=1}^K\int_0^T[\tilde{y}_k(t)-d_k(t)]\frac{\partial\tilde{y}_k(t)}{\partial\alpha^y_{ij}}dt \quad (92)$$

where $\frac{\partial\tilde{y}_k(t)}{\partial\alpha^y_{ij}}$ is given by

$$\begin{aligned}\frac{\partial\tilde{y}_k(t)}{\partial\alpha^y_{ij}} &= \frac{\partial\dfrac{\sum_{i=1}^n\mathcal{E}^y_i(t)\prod_{g=1}^p S(u_{kg}(t),\mathcal{E}^x_{ig}(t))}{\sum_{i=1}^n\prod_{g=1}^p S(u_{kg}(t),\mathcal{E}^x_{ig}(t))}}{\partial\alpha^y_{ij}} \\ &= \frac{\prod_{g=1}^p S(u_{kg}(t),\mathcal{E}^x_{ig}(t))}{\sum_{i=1}^n\prod_{g=1}^p S(u_{kg}(t),\mathcal{E}^x_{ig}(t))}\frac{\mathcal{E}^y_i(t)}{\partial\alpha^y_{ij}} \\ &= \frac{\widetilde{\mathcal{E}}^y_j(t)\prod_{g=1}^p S(u_{kg}(t),\mathcal{E}^x_{ig}(t))}{\sum_{i=1}^n\prod_{g=1}^p S(u_{kg}(t),\mathcal{E}^x_{ig}(t))}. \end{aligned} \quad (93)$$

VIII. Concluding Remarks

In this paper I only presented a few possible learning algorithms of computational verb rules based on known samples. The computational verb rules discussed include the following cases.

1) The antecedent of each computational verb rule consists of one computational verb and the consequent consists of one real number;

$$
\begin{aligned}
\frac{\partial \tilde{y}_k(t)}{\partial \alpha_{ij}^{(hx)}} &= \frac{\partial \dfrac{\sum_{i=1}^{n} \mathcal{E}_i^y(t) \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))}{\sum_{i=1}^{n} \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))}}{\partial \alpha_{ij}^{(hx)}} \\
&= \prod_{g=1, g\neq h}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t)) \left\{ \frac{\mathcal{E}_i^y(t)}{\sum_{i=1}^{n} \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))} - \frac{\sum_{i=1}^{n} \mathcal{E}_i^y(t) \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))}{\left[\sum_{i=1}^{n} \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))\right]^2} \right\} \frac{\partial S(u_{kh}(t), \mathcal{E}_{ih}^x(t))}{\partial \alpha_{ij}^{(hx)}} \\
&= \prod_{g=1, g\neq h}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t)) \left\{ \frac{\mathcal{E}_i^y(t)}{\sum_{i=1}^{n} \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))} - \frac{\sum_{i=1}^{n} \mathcal{E}_i^y(t) \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))}{\left[\sum_{i=1}^{n} \prod_{g=1}^{p} S(u_{kg}(t), \mathcal{E}_{ig}^x(t))\right]^2} \right\} \\
&\quad \times \frac{\int_0^T \left[s_t\left(u_{kh}(t)\right) - s_t\left(\sum_{j=1}^{m_x} \alpha_{ij}^{(hx)} \widetilde{\mathcal{E}}_j^x(t) \right) \right] \dot{s}_t\left(\sum_{j=1}^{m_x} \alpha_{ij}^{(hx)} \widetilde{\mathcal{E}}_j^x(t) \right) \widetilde{\mathcal{E}}_j^x(t) dt}{T \sqrt{\frac{1}{T} \int_0^T \left[s_t(u_{kh}(t)) - s_t\left(\sum_{j=1}^{m_x} \alpha_{ij}^{(hx)} \widetilde{\mathcal{E}}_j^x(t) \right) \right]^2 dt}}. \qquad (91)
\end{aligned}
$$

2) The antecedent of each computational verb rule consists of more than one computational verb and the consequent consists of one real number;
3) The antecedent of each computational verb rule consists of one real number and the consequent consists of one computational verb;
4) Fuzzification of 3);
5) The antecedent of each computational verb rule consists of more than one real number and the consequent consists of one computational verb.
6) Fuzzification of 5);
7) The antecedent and consequent of each rule consists of one computational verb;
8) The antecedent consists of more than one computational verbs and the consequent consists of one computational verb.

There are many ways to generalize the results presented here. First, all real numbers can be generalized into fuzzy membership functions. Second, the way to calculate computational verb similarities can be generalized into many others reported in [36] as well. Third, the types of computational verb rules can be generalized into many others. Fourth, the ways of constructing computational verbs used in computational verb rules may have many different choices. Therefore, the results presented here is far from complete or comprehensive.

REFERENCES

[1] Guanrong Chen and Trung Tat Pham. *Introduction to Fuzzy Systems*. Chapman & Hall/CRC, November 2005. ISBN:1-58488-531-9.

[2] Yang's Scientific Research Institute LLC. **FireEye** *Visual Flame Detecting Systems*. http://www.yangsky.us/products/flamesky/index.htm, http://www.yangsky.com/products/flamesky/index.htm, 2005.

[3] Yang's Scientific Research Institute LLC. **BarSeer** *Webcam Barcode Scanner*. http://www.yangsky.us/demos/barseer/barseer.htm, http://www.yangsky.com/demos/barseer/barseer.htm, 2006.

[4] Yang's Scientific Research Institute LLC. *Cognitive Stock Charts*. http://www.yangsky.us/products/stock/, http://www.yangsky.com/products/stock/, 2006.

[5] Yang's Scientific Research Institute LLC. **PornSeer** *Pornographic Image and Video Detection Systems*. http://www.yangsky.us/products/dshowseer/porndetection/PornSeePro.htm, http://www.yangsky.com/products/dshowseer/porndetection/PornSeePro.htm, 2006.

[6] Yang's Scientific Research Institute LLC. and Wuxi Xingcard Technology Ltd. **YangSky-MAGIC** *Visual Card Counters*. http://www.yangsky.us/products/cardsky/cardsky.htm, http://www.yangsky.com/products/cardsky/cardsky.htm, 2004.

[7] Yang's Scientific Research Institute LLC. and Chinese Traffic Management Research Institute of the Ministry of Public Security(TMRI-China). **DriveQfy** *Automatic CCTV Driver Qualify Testing Systems*. http://www.yangsky.us/products/driveqfy/driveqfy.htm, http://www.yangsky.com/products/driveqfy/driveqfy.htm, 2005.

[8] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in* C*: The Art of Scientific Computing(2nd Edition)*. Cambridge University Press, October 30, 1992.

[9] T. Yang. Verbal paradigms—Part I: Modeling with verbs. Technical Report Memorandum No. UCB/ERL M97/64, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, 9 Sept. 1997. page 1-15.

[10] T. Yang. Verbal paradigms—Part II: Computing with verbs. Technical Report Memorandum No. UCB/ERL M97/66, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, 18 Sept. 1997. page 1-27.

[11] T. Yang. Computational verb systems: Computing with verbs and applications. *International Journal of General Systems*, 28(1):1–36, 1999.

[12] T. Yang. Computational verb systems: Adverbs and adverbials as modifiers of verbs. *Information Sciences*, 121(1-2):39–60, Dec. 1999.

[13] T. Yang. Computational verb systems: Modeling with verbs and applications. *Information Sciences*, 117(3-4):147–175, Aug. 1999.

[14] T. Yang. Computational verb systems: Verb logic. *International Journal of Intelligent Systems*, 14(11):1071–1087, Nov. 1999.

[15] T. Yang. Computational verb systems: A new paradigm for artificial intelligence. *Information Sciences—An International Journal*, 124(1-4):103–123, 2000.

[16] T. Yang. Computational verb systems: Verb predictions and their applications. *International Journal of Intelligent Systems*, 15(11):1087–1102, Nov. 2000.

[17] T. Yang. Computational verb systems: Verb sets. *International Journal of General Systems*, 20(6):941–964, 2000.

[18] T. Yang. *Advances in Computational Verb Systems*. Nova Science Publishers, Inc., Huntington, NY, May 2001. ISBN 1-56072-971-6.

[19] T. Yang. Computational verb systems: Computing with perceptions of dynamics. *Information Sciences*, 134(1-4):167–248, Jun. 2001.

[20] T. Yang. Computational verb systems: The paradox of the liar. *International Journal of Intelligent Systems*, 16(9):1053–1067, Sept. 2001.

[21] T. Yang. Computational verb systems: Verb numbers. *International Journal of Intelligent Systems*, 16(5):655–678, May 2001.

[22] T. Yang. *Impulsive Control Theory*, volume 272 of *Lecture Notes in Control and Information Sciences*. Spinger-Verlag, Berlin, Aug. 2001. ISBN 354042296X.

[23] T. Yang. *Computational Verb Theory: From Engineering, Dynamic Systems to Physical Linguistics*, volume 2 of *YangSky.com Monographs in Information Sciences*. Yang's Scientific Research Institute, Tucson, AZ, Oct. 2002. ISBN:0-9721212-1-8.

[24] T. Yang. Computational verb systems: Verbs and dynamic systems. *International Journal of Computational Cognition*, 1(3):1–50, Sept. 2003.

[25] T. Yang. *Fuzzy Dynamic Systems and Computational Verbs Represented by Fuzzy Mathematics*, volume 3 of *YangSky.com Monographs in Information Sciences*. Yang's Scientific Press, Tucson, AZ, Sept. 2003. ISBN:0-9721212-2-6.

[26] T. Yang. *Physical Linguistics: Measurable Linguistics and Duality Between Universe and Cognition*, volume 5 of *YangSky.com Monographs in Information Sciences*. Yang's Scientific Press, Tucson, AZ, Dec. 2004.

[27] T. Yang. Simulating human cognition using computational verb theory. *Journal of Shanghai University(Natural Sciences)*, 10(s):133–142, Oct. 2004.

[28] T. Yang. Architectures of computational verb controllers: Towards a new paradigm of intelligent control. *International Journal of Computational Cognition*, 3(2):74–101, June 2005 [available online at $http://www.YangSky.com/ijcc/ijcc32.htm$, $http://www.YangSky.us/ijcc/ijcc32.htm$].

[29] T. Yang. Applications of computational verbs to the design of P-controllers. *International Journal of Computational Cognition*, 3(2):52–60, June 2005 [available online at $http://www.YangSky.us/ijcc/ijcc32.htm$, $http://www.YangSky.com/ijcc/ijcc32.htm$].

[30] T. Yang. Applications of computational verbs to digital image processing. *International Journal of Computational Cognition*, 3(3):31–40, September 2005 [available online at $http://www.YangSky.us/ijcc/ijcc33.htm$, $http://www.YangSky.com/ijcc/ijcc33.htm$].

[31] T. Yang. Bridging the Universe and the Cognition. *International Journal of Computational Cognition*, 3(4):1–15, December 2005 [available online at $http://www.YangSky.us/ijcc/ijcc34.htm$, $http://www.YangSky.com/ijcc/ijcc34.htm$].

[32] T. Yang. Applications of computational verbs to effective and realtime image understanding. *International Journal of Computational Cognition*, 4(1):49–67, March 2006 [available online at $http://www.YangSky.com/ijcc/ijcc41.htm$, $http://www.YangSky.us/ijcc/ijcc41.htm$].

[33] T. Yang. Applications of computational verbs to feeling retrieval from texts. *International Journal of Computational Cognition*, 4(3):28–45, September 2006 [available online at $http://www.YangSky.com/ijcc/ijcc43.htm$, $http://www.YangSky.us/ijcc/ijcc43.htm$].

[34] T. Yang. Applications of computational verbs to cognitive models of stock markets. *International Journal of Computational Cognition*, 4(2):1–13, June 2006 [available online at $http://www.YangSky.us/ijcc/ijcc42.htm$, $http://www.YangSky.com/ijcc/ijcc42.htm$].

[35] T. Yang. Applications of computational verbs to the study of the effects of Russell's annual index reconstitution on stock markets. *International Journal of Computational Cognition*, 4(3):1–8, September 2006 [available online at $http://www.YangSky.us/ijcc/ijcc43.htm$, $http://www.YangSky.com/ijcc/ijcc43.htm$].

[36] T. Yang. Distances and similarities of saturated computational verbs. *International Journal of Computational Cognition*, 4(4):62–77, December 2006 [available online at $http://www.YangSky.us/ijcc/ijcc44.htm$, $http://www.YangSky.com/ijcc/ijcc44.htm$].

[37] T. Yang. Using computational verbs to cluster trajectories and curves. *International Journal of Computational Cognition*, 4(4):78–87, December 2006 [available online at $http://www.YangSky.us/ijcc/ijcc44.htm$, $http://www.YangSky.com/ijcc/ijcc44.htm$].

[38] T. Yang. Accurate video flame-detecting system based on computational verb theory. AS *Installer*, (42):154–157, August 2007. (in Chinese).

[39] T. Yang. Applications of computational verb theory to the design of accurate video flame-detecting systems. *International Journal of Computational Cognition*, 5(3):25–42, September 2007 [available online at $http://www.YangSky.us/ijcc/ijcc53.htm$, $http://www.YangSky.com/ijcc/ijcc53.htm$].

[40] Jian Zhang and Minrui Fei. Determination of verb similarity in computational verb theory. *International Journal of Computational Cognition*, 3(3):74–77, September 2005 [available online at $http://www.YangSky.us/ijcc/ijcc33.htm$, $http://www.YangSky.com/ijcc/ijcc33.htm$].