A Developmental Method that Computes Optimal Networks without Post-Selections

Juyang Weng*†‡§
*Department of Computer Science and Engineering, †Cognitive Science Program, ‡Neuroscience Program,
Michigan State University, East Lansing, MI, 48824 USA
§GENISAMA LLC, Okemos, MI 48864 USA

***Abstract*—This work is the theory of Post-Selection practices that have been rarely studied. Post-Selections mean selections of systems after the systems have been trained. Post-Selections Using Validation Sets (PSUVS) are wasteful and Post-Selections Using Test Sets (PSUTS) are wasteful and unethical. Both result in systems whose generalization powers are weak. The PSUTS fall into two kinds, machine PSUTS and human PSUTS. The connectionist AI school received criticisms for its "scruffiness" due to a huge number of network parameters and now the machine PSUTS; but the seemingly "clean" symbolic AI school seems more brittle because of its human PSUTS. This paper analyzes why, in deep learning, error-backprop methods with random initial weights suffer from severe local minima, why PSUTS violates well-established research ethics, and publications that used PSUTS should have transparently reported PSUTS. This paper proposes a Developmental Methodology that trains only a single but optimal network for each application lifetime using a new standard for performance evaluation in machine learning, called developmental errors for all networks trained in a project that the selection of the luckiest network depends on, along with Three Learning Conditions: (1) framework restrictions, (2) training experience and (3) computational resources. This paper also discusses how the brain-inspired Developmental Networks (DNs) avoid PSUTS by reporting developmental errors and its maximum likelihood (ML) optimality under the Three Learning Conditions. DNs are not "scruffy" because they are ML-estimators of the observed Emergent Turing Machines at each time during their "lives". This implies best performance given a limited amount of overall available computational resources for a project.**

I. Introduction

In 2000, the author published with six (6) co-authors what is called "Autonomous Mental Development (AMD)" in *Science* [1]. The *Science* editor commented that developmental AI had not been not known before—the article published a new direction "task-nonspecific programs across lifetime". All programs before then were task-specific, including all published neural networks due to PSUTS to be explained below. A series of advances has been made since 2000. The following is only an outline of the most important events. For a more detailed account of entire picture, the reader is referred to a book by the author [2].

In 2017, the author and his coworkers presented "emergent universal Turing machine" [3]. This is the first emergent mode about Auto-Programming for General Purposes (APFGP) along with experimental results.

Last year, 2020, the author argued APFGP enables conscious machines, the first model about conscious machines that are based on emergent universal Turing machines [4].

Here, this author presented a controversial practice called Post-Selection Using Test Sets (PSUTS). This report also explains why the author's brain model (Developmental Networks, DNs) avoids PSUTS. Using PSUTS means the corresponding project wastes much resource of computation and manpower and the superficial error of the reported system is misleading because the system does not give a similar error on new data sets. This paper not only raised a controversy but also presented a solution to avoid PSUTS.

Since 2012, AI has attracted much attention from public and media. A large number of projects in AI has published. If the authors of these projects use the method of this report, they could benefit much for reducing the time and manpower used to reach their target systems as well as improving the generalization powers of their target systems.

Let us first discuss in Sec. II what PSUTS is. Sec. III reasons why error backprop needs PSUTS. Sec. IV explains why Developmental Networks (DNs) do not need and avoid PUSTS. Sec. V provides details of the methodology. Sec. VI outlines experiments. Sec. VII presents concluding remarks.

II. Post-Selections

Many machine-learning methods were evaluated without considering how much computational recourses are necessary for the development of a reported system. Thus, comparisons about the performance of the system have been biased toward a competition of how much resources a group has, as we will see below after we define the Post-Selection, regardless how many networks have been trained and discarded, and how large each network is. Worse still, test sets were used in a controversial way. Here we explicitly define a set of Three Learning Conditions for development of an AI system:

The Three Learning Conditions for developing an AI system are: (1) the framework restrictions, including whether task-specific or task-nonspecific, batch learning or incremental learning, and the body's sensors and effectors; (2) the teaching experience; (3) the computational resources including the number of hidden neurons.

A. Post-Selections

The available data set D is divided into three mutually disjoint sets, a training set T, a validation set V, and a test set

J. Weng, "Developmental Method that Computes Optimal Networks without Post-Selections," in Proc. IEEE International Conference on Development and Learning, pp. 1-6, Beijing, China, Aug. 23-26, 2021.

T'. Two sets are disjoint if they do not share any elements. Let us consider how the validation sets and test sets are used in experiments.

A network architecture has a set of parameters represented by a vector, where each component corresponds to an architecture parameter, such as convolution kernel sizes and stride values at each level of a deep hierarchy, the neuronal learning rate, and the neuronal learning momentum value, etc. Let k be a finite number of grid points along which such architecture parameter vectors need to be tried, $A = \{\mathbf{a}_i \mid i = 1, 2, ..., k\}$. If there are 10 parameters in each architecture and each of which has 10 grid points to try, there are a total of $k = 10^{10} = 10$B architecture parameter vectors to try, an extremely large number.

For each architecture vector $\mathbf{a}_i$, assume n sets of random weights $\mathbf{w}_j$, resulting in kn networks

$$\{N(\mathbf{a}_i, \mathbf{w}_j) \mid i = 1, 2, ..., k, j = 1, 2, ..., n\}$$

are trained each of which starts with a different set of random weights $\mathbf{w}_j$, using error backprop that locally and numerically minimizes the fitting error $f_{i,j}$ on the training set T. Graves et al. [5] seems to have mentioned $n = 20$. Using the above example of $k = 10B$, $kn = 200$B a huge number that requires a lot of computational resources and manpower.

Let us define the Post-Selection. Suppose that the trainer is first aware of the validation sets and the test sets. He trains multiple systems using the training sets. After these systems have been trained, he post-selects a system by searching, manually or assisted by computers, among trained systems based on the validation sets or the test sets. This is called Post-Selection—selection after training.

Obviously, a post-selection wastes all trained systems except the selected one. As we will see next, a system from the post-selection has a weak generalization power.

First, consider Post-Selections Using Validation Sets:

B. PSUVS

A Machine PSUVS is defined as follows: If the test set T' is not available, suppose the validation error of $N(\mathbf{a}_i, \mathbf{w}_j)$ is $e_{i,j}$ on the validation set V, find the best network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches the minimum validation error:

$$e_{i^*,j^*} = \min_{1 \le i \le k, 1 \le j \le n} e_{i,j} \tag{1}$$

and report only the performance e_{i^*,j^*} but not the performances of other remaining $kn - 1$ trained neural networks.

This is obviously a violation of the principle of cross-validation. Because each vector $\mathbf{a}_i$ uses n random weights, an average of $e_{i,j}$ over n for each $\mathbf{a}_i$ should be used instead of the minimum in Eq. (1) if we want to fairly eliminate lucks from the result. This leads to

$$\bar{e}_{i^*,j^*} = \min_{1 \le i \le k} \frac{1}{n} \sum_{j=1}^{n} e_{i,j}. \tag{2}$$

Similarly, a Human PSUVS is one by which a human selects a system from multiple trained systems using their validation errors.

Since a PSUVS procedure picks the best system based on the errors on the validation set, the resulting system might not do well on the test sets because doing well on validation sets do not guarantee to do well on the test sets.

Worse is Post-Selections Use Test Set (PSUTS). There are two kinds of PSUTS, machine PSUTS and human PSUTS.

C. Machine PSUTS

If the test set T' is available which seems to be true for almost all neural network publications, we define Post-Selection Using Test Sets (PSUTS):

A Machine PSUTS is defined as follows: If the test set T' is available, suppose the test error of $N(\mathbf{a}_i, \mathbf{w}_j)$ is $e'_{i,j}$ on the test set T', find the best network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches the minimum test error:

$$e'_{i^*,j^*} = \min_{1 \le i \le k, 1 \le j \le n} e'_{i,j} \tag{3}$$

and report only the performance e'_{i^*,j^*} but not the performances of other remaining $kn - 1$ trained neural networks.

Imagine that we want to remove lucks in the above expression, by using averages like we did in Eq. (2):

$$\bar{e}'_{i^*,j^*} = \min_{1 \le i \le k} \frac{1}{n} \sum_{j=1}^{n} e'_{i,j}. \tag{4}$$

But the above error is still unethical and misleading since each term under minimization for $\mathbf{a}_i$ has peeked into test sets!

There are some variations of Machine PSUTS: The validation set V or T' are not disjoint with T. If $T = V$, we call it validation-vanished PSUTS. If $T = T'$, we called it test-vanished PSUTS.

A distribution of fitting errors, validation errors and test errors is defined as follows: The distributions of all kn trained networks' fitting errors $\{f_{ij}\}$, validation errors $\{e_{ij}\}$, and test errors $\{e'_{ij}\}$, $i = 1, 2, ...k$, $j = 1, 2, ...n$, as well as the values of k and n.

It is necessary to present some key statistical characteristics of such distributions. For example, ranked errors in decreasing order. Then given the maximum, 75%, 50%, 25%, and minimum value of these kn values for the fitting errors, validation errors. and test errors, so that the research community can see whether error-backprop can avoid local minima in deep learning. Unfortunately, the author did not find such reports. Our experience in experiments indicated that the maximum and the minimum values of the distribution of fitting errors alone are drastically different, around 80% and 5%, respectively. Section III will discuss why.

Further, such a use test sets to post-select networks resembles hiring a larger number kn of random test takers and report only the luckiest $N(\mathbf{a}_{i*}, \mathbf{w}_{j^*})$ after the answer-based grading. This practice could hardly be acceptable to any test agencies and any agencies that will use the test scores for admission since this submitted error e'_{i^*,j^*} misleads due to its lack of generalization.

The architecture parameter vector $\mathbf{a}_{i^*}$ and weights $\mathbf{w}_{j^*}$ overfits T, V and T'. If an unobserved data set T'', disjoint with

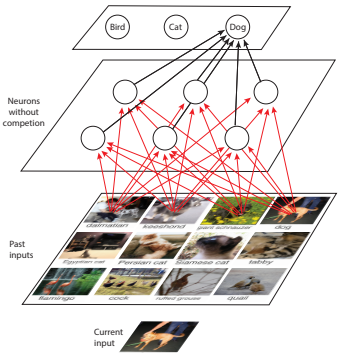

Fig. 1. Lack of role-determination in hidden neurons due to a lack of competition. The same ideas are true for a deeper hierarchy. Color sample images courtesy of [8].

T', $T' \cap T'' = \emptyset$, is observed from a new environment, the error rate e''_{i^*,j^*} of $N(\mathbf{a}_{i*}, \mathbf{w}_{j^*})$ is predicted to significantly higher than e'_{i^*,j^*},

$$e''_{i^*,j^*} \gg e'_{i^*,j^*} \tag{5}$$

because Eq. (3) depends on the test set T' in the Post-Selection from many networks.

Weng 2020 argued that some public speakers claimed that such misleading errors have approached or even succeeded human performance seem to be not only questionable but also controversial since the test sets seem to have been used without sufficient disclosure.

D. Human PSUTS

Instead of writing a search program in Machine PSUTS, Human PSUTS defined below typically involves less computational resources and programming demands.

A human PSUTS is defined as follows: After planning or knowing what will be in the training set T and test set T', a human post-selects features in networks instead of using a machine to learn such features.

Unfortunately, almost all methods in the symbolic school use Human PSUTS because it is always the same human who plans for and design a micro-world (e.g., handcrafting a set of task-specific symbols [6] using graphic neural networks) and collect the test set T' only within the micro-world. A critical criterion for an acceptable test score lies in whether a learned machine goes beyond the micro-world—a key capability of conscious learning [4] by avoiding using symbols.

III. Why Error Backprop Requires PSUTS

Weng 2021 [7] used mathematical analysis to prove the following theorem.

Theorem 1 (Lacks of Error-BackProp): Error-backprop lacks (1) energy conservation, (2) an age-dependent learning rate, and (3) competition based role-determination.

The meaning (3) of Theorem 1 are illustrated by Fig. 1. CNNs do not have a competition mechanism in hidden layers. Complete connections initialized with random weights are provided for all consecutive areas (also called layers), from input area all the way to the output area. If the z_j neuron is in the output motor area and each output neuron is assigned a single class label, the role of z_j ("dog" in the figure) is determined by human supervised label "dog". However, let us assume instead that z_j is in a hidden area, not responsible for the "dog" class. z_j still updates its input weights using the gradient. Likewise, the pre-synaptic area Y, is labeled "neurons without competition". The hidden neurons in this area do not have a competition mechanism to decide the role of each neuron there.

For further detail about why error-backprop requires PSUTS, the reader is referred to Sec. III of [7].

IV. How DNs Avoid PUSTS

Apparently, brains do not use PUSTS, as every human child must normally develop in a human environment to make the living. Cresceptron [9] and later DN [10], [3] were inspired by the interactive mode that brains learn though lifetime. Namely, every DN must be ML-optimal given the same Three Learning Conditions.

A. Developmental Errors

In contrast to PSUTS, we define and reported developmental errors that includes all errors occurred through lifetime of each learning network:

A developmental error is defined as follows: The developmental errors of a developmental network $N = (X, Y, Z, M)$ with sensory area X, skull closed hidden area Y and motor area Z and memory M, runs through lifetime by sampling at discrete time indices $t = 0, 1, 2, ...$ as $N(t) = (X(t), Y(t), Z(t), M(t))$. Start at inception $t = 0$ with supervised sensory input $\mathbf{x}_0 \in X(0)$, initial state $\mathbf{z}_0 \in Z(0)$, and randomly initialized weigh vector $\mathbf{y}_0 \in Y(0)$, and initial memory $\mathbf{m}_0 \in M(0)$. At each time $t = 1, 2, ...$, the network N recursively and incrementally updates:

$$(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{m}_t) = f(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1}, \mathbf{m}_{t-1}) \tag{6}$$

where f is the Developmental Program (DP) of N. If $\mathbf{z}_t \in Z(t)$ is supervised by the teacher, the network complies and the error e_t is recorded, but if the supervised motor vector has error, the error should be treated as teacher's. Otherwise, the learner is not motor-supervised and N generates a motor vector $\mathbf{z}_t$ and is observed by the teacher and its deviation from the desired $\mathbf{z}_t^*$ is recorded as error e_t. The lifetime average error from time 0 to time t is defined as

$$\bar{e}(t) \triangleq \frac{1}{t} \sum_{i=0}^{t} e_i. \tag{7}$$

Namely, the developmental error, unless stated otherwise for a particular time period, is the average lifetime error. For more detailed information about the process of errors $\{\mathbf{e}_t \mid t \geq 0\}$, other statistical characterizations can be utilized, such as standard deviation, variance, and ranked statistics such as minimum, 25%, 50% (median), 75%, and maximum errors.

Because Cresceptron and DN have a dynamic number of neurons up to a system memory limit, each new context

$$\mathbf{c}_t \triangleq (\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \tag{8}$$

may be significantly different from the nearest matched learned weight vectors of all hidden neurons. If that happens and there are still new hidden neuron that have not fired, a new hidden neuron is spawned that perfectly memorize this new context regardless its randomly initialized weights. When all the available hidden neurons have fired at least once, the DN will update the top-k matched neurons optimally in the sense of maximum likelihood (ML), as proven for DN-1 by [11] and for DN-2 by [12]. For more specific time periods, such as the period from time t_1 to t_2 during which only disjoint tests were made by the teacher and the learning agent is not motor-supervised, the average error is denoted as $\bar{e}(t_1 : t_2)$. Therefore, $\bar{e}(t)$ means $\bar{e}(0 : t)$.

Note that a developmental system has two input areas from the environment, sensory X and motor Z. Since there is hardly any sensory input $\mathbf{x} \in X$ that exactly duplicates at two different time indices, almost all sensory inputs from X are sensory-disjoint. During motor-supervised learning, if the teacher supervises its motor area Z and the learner complies. Since a teacher can take an error, the motor-error that the teacher made is also recorded as motor error from the learner but due to the teacher.

B. Competition

As discussed above, error backprop learning is without completion. The main purpose of competition is to automatically assign roles among hidden neurons. Below, we consider two cases, sensory networks that are simpler and sensorimotor networks which are more complex but much more powerful and brain-like.

1) Sensory networks: Let us first consider the case of feed-forward networks as illustrated in Fig. 2. Fig. 2(a) shows a situation where the number of samples in X is larger than the number of hidden neurons, which is typical and natural. Otherwise, if there are sufficient hidden neurons, each hidden neuron can simply memorize a single sample $\mathbf{x} \in X$.

This means that the total number of hidden neurons must be shared through incremental learning, where each sample image-label pair $(\mathbf{x}, s) \in X \times S$ arrives incrementally through time, $t = 0, 1, 2, ...$ This is the case with Cresceptron which conducts incremental learning by dealing with image-label pairs one at a time and update incrementally.

Every layer in Cresceptron consists of a image-feature kernel, which is very different from those in DN where each hidden neuron represents a sensorimotor feature to be discussed later. By image-feature, we mean that each hidden neuron is centered at an image pixel. Competitions take place within the column for a receptive field centered at each pixel at the resolution of the layer. The resolution reduces from lower layer to higher layer through was called resolution reduction (drop-out).

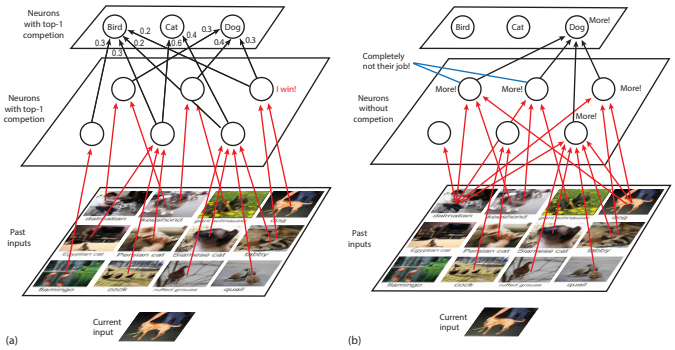

Fig. 2. How competition automatically assigns roles among hidden neurons without a central controller: The case for automatically construct a mapping $f : X \mapsto S$. (a) The number of samples in X is larger than the number of hidden neurons such that each hidden neuron must win-and-fire for multiple inputs. (b) Error-backprop from the "dog" motor neuron asks some hidden neurons to help but the current input feature is not their job. Thus, error-backprop messes up with the role assignment guessed by the random initial weights. The same ideas are true for a deeper hierarchy. Color sample images courtesy of [8].

The competition in incremental learning is represented by incrementally assigning a new neuronal plane (convolution plane) where the new kernel memorizes the new input pattern if the best matched neuron in a column does not match sufficiently well. Suppose images $\mathbf{x} \in X$ arrives sequentially, the top-1 competition in the hidden layer in Fig. 2(a) enables each hidden neuron to respond to multiple features, indicated by the typically multiple upward arrows, one from each image, pointing to a hidden neuron. This amounts to incremental clustering based on top-k competition. The weight vector of each hidden Y neuron corresponds to a cluster in the X space. In Fig. 2(a), $k = 1$ for top-k competition in Y.

Likewise, suppose top-1 competition in the next higher layer, say Z, namely each time only one Z neuron is supervised to fire at 1 and all other Z neurons do not fire, resulting the connection patterns from the second layer Y to the next higher layer Z.

The Candid Covariance-free Incremental (CCI) Lobe Component Analysis (LCA) in Weng 2009 [13] proved that such automatic assignment of roles through competition results in a dually optimal neuronal layer, optimal spatially and optimal temporally. Optimal spatially means the CCI LCA incrementally computes the first principal component features of the receptive field. Optimal temporally means that the principal component vector has the least expected distance to its target—the optimal estimator in the sense of minimum variance to the true principal component vector.

Intuitively, regardless what random weights each hidden neuron starts with, as soon as it is spawn to fire, its firing age $a = 1$. Its random weight vector is multiplied by the zero retention rate $w_1 = 1 - 1/a = 0$ and this learning rate $w_2 = s1/a = 1$ so that the new weight vector before becomes the first input $r\mathbf{x}$ with $r = 1$ for the firing winner.

$$\mathbf{v} \leftarrow (1 - \frac{1}{a})\mathbf{v} + \frac{1}{a} r\mathbf{x}. \tag{9}$$

It has been proven that the above expression incrementally computes the first principal component as $\mathbf{v}$. The learning rate $w_2 = \frac{1}{a}$ is the optimal and age-dependent learning rate. CCI LCA is a framework for dually optimal Hebbian learning. The property "candid" corresponds to the property that sum of the learning rate $w_2 = \frac{1}{a}$ and the retention rate $w_1 = 1 - \frac{1}{a}$ is always 1 to keep the "energy" of response r weighted input $\mathbf{x}$ unchanged (e.g., not to explode or vanish). This dually optimality resolves the three problems in Theorem 1.

Fig. 2(b) shows how the three neurons in the Z area updates their weights so that the weight from the second area to the third area become the probability of firing, conditioned on the firing of the post-synaptic neuron in area Z (Dog, Cat, Bird, etc.). The CCI LAC guarantees that the sum of weights for each Z neuron sum to 1. This automatic role assignment optimally solves the random roles in error-backprop established by Theorem 3 of [7].

However, optimal network for incrementally constructing a mapping $f : X \mapsto S$ is too restricted, since $f : X \mapsto S$ is only what brains can do, but not all brains can do. For the latter, we must address sensorimotor networks.

2) Sensorimotor networks: The main reason that Marvin Minsky [14] complained that neural network is scruffy was because conventional neural networks lacked not only the optimality described above for sensory networks, but also lacked the Emergent Universal Turing Machines (EUTM) that is ML-optimal we now discuss below.

First, each neuron in the brain not only corresponds to a sensory feature as illustrated in Fig. 2, but also a sensorimotor feature. By sensorimotor feature, we mean that the firing of each hidden neuron in Fig. 2 is determined not just by the current image σ represented by a sensory vector $\mathbf{x} \in X$, but also the state q represented by a motor vector $\mathbf{z} \in Z$. It is well known that a biological brain contains not only bottom-up inputs from $\mathbf{x} \in X$ but also top-down inputs from $\mathbf{z} \in Z$. In summary, each hidden neuron represents a sensorimotor feature in a complex brain-like network.

C. FAs as sensorimotor mappings

The sensorimotor features are easier to understand if we use symbols. Let us borrow the idea of Finite Automaton (FA). In an FA, transitions are represented by function $\delta : Q \times \Sigma \mapsto Q$, where Σ is the set of input symbols and Q the set of states. Each transition is represented by

$$(q, \sigma) \xrightarrow{f} q'$$

a) AFA as a control of any Turing machine: Weng 2015 [11] extended the definition an FA so that it outputs its state so the resulting FA becomes an Agent FA (AFA). Further, Weng 2015 [11] extended the action q to the machinery of Turing machine (see Fig. 3 so that action q includes output symbol to the Turing tape and the head motion of the read-write head of a Turing machine. With this extension, Weng 2015 [11] proved that any Turing machine is an AFA.

Here $q \in Q$ is the top-down motor input to a sensorimotor feature neuron; σ is the bottom-up sensory input to the same neuron. If δ has n transitions, n hidden neurons in the Y area are sufficient to memorize all the transitions that is observed sequentially, one transition at a time.

Since we should not use symbols like σ and q, but sensory vectors $\mathbf{x} \in X$ and motor vectors $\mathbf{z} \in Z$. At discrete time $t = 0, 1, 2, ...$, we use the hidden neurons in the Y area to incrementally learn the transitions:

$$\begin{bmatrix} Z(0) \\ Y(0) \\ X(0) \end{bmatrix} \rightarrow \begin{bmatrix} Z(1) \\ Y(1) \\ X(1) \end{bmatrix} \rightarrow \begin{bmatrix} Z(2) \\ Y(2) \\ X(2) \end{bmatrix} \rightarrow ... \qquad (10)$$

where $\rightarrow$ means neurons on the right use the input neurons on the right and compete to fire as explained below without iterations. Namely, by unfolding time, the spatially recurrent DN becomes non-recurrent in a time-unfolded and time-sampled DN that is ML-optimal and has a large constant complexity $O(1)$ as proven in [11],suited for real-time computation with a large memory and many parallel computing elements.

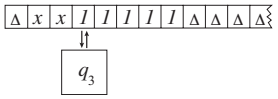

Fig. 3. A Turing machine has a tape, a read-write head, and a transition function with a current state.

D. DNs as ML-Optimal Universal Super-Turing Machines

The traditional Turing Machine (TM) is a human handcrafted machine, as illustrated in Fig. 3.

A Universal TM (UTM) s still a TM, but its tape contains two parts, a user supplied program and the data that the program is applied to. The transition function of the UTM is designed to simulate any program encoded in the form of transition of a TM and to apply the program on the data on the tape and finally to place the output of the program on the data onto the tape.

A UTM is a model for practical computers in commercial markets because the user can write any program on any set of appropriate data for the UTM to carry out. Because a DN is an ML-optimal emergent FA, Weng 2015 [11] extended a symbolic Turing machine to a super Turing machine by (1) extending the tape to the real world, (2) the input symbols to vectors from sensors, (3) the output symbols to vector output from effectors, and (3) the head motion to any action from the agent. Thus, DN ML-optimally learns any TM, including UTM, directly from the physical world.

E. DNs as ML-Optimal for APFGP

Because DN is an ML-optimal learning engine for any TM, including UTM, DN ML-optimally learns any UTM from the physical world, conditioned on the Three Learning Conditions. This means that a DN ML-optimally learns to Autonomous Programming for General Purposes (APFGP) [3]. Based on the capability of APFGP, Weng 2020 argued that APFGP is a characterization of conscious machines [4] that boots its skill of consciousness through conscious learning—being

conscious while learning across lifetime. Hopefully, APFGP is a clearer and more precise characterization for conscious machines and animals, assuming that we allow a conscious machine to develop its consciousness from infancy.

In the following, we present details in Methods.

V. Methods

Given the Three Learning Conditions, at each time t, $t = 1, 2, ...$, a DN incrementally computes the ML-estimator of its parameters at each time t that minimizes the developmental error without doing any iterations.

Let us first review the maximum likelihood estimator for a batch data. Let $\mathbf{x}$ be the observed data and $f_\theta(\mathbf{x}, \mathbf{z})$ is the probability density function that depends on a vector θ of parameters. The maximum estimator for θ corresponds to the θ that maximizes the probability density, there θ contains all parameters in the network, including the hidden Y area. Regardless $\mathbf{z}$ is imposed, $\mathbf{z}$ is part of the parameters to be computed as self-generated version:

$$(\theta^*, \mathbf{y}^*, \mathbf{z}^*) = \underset{(\theta, \mathbf{y}, \mathbf{z})}{\operatorname{argmax}} f_\theta(\mathbf{x}). \tag{11}$$

Since lifetime estimator is incremental, at each time t, the previous state $\mathbf{z}^*_{t-1}$ is self-generated or supervised, and the observation is $\mathbf{x}_{t-1}$. The incremental ML-estimator for θ^*_t is computed by the incremental version of Eq. (11) where f uses context $\mathbf{c}_{t-1} = (\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1})$:

$$(\theta^*_t, \mathbf{y}^*_t, \mathbf{z}^*_t) = \underset{(\theta_t, \mathbf{y}_t, \mathbf{z}_t)}{\operatorname{argmax}} f_{\theta_t}(\mathbf{x}_{t-1}, \mathbf{y}^*_{t-1}, \mathbf{z}^*_{t-1}). \tag{12}$$

The DN computes the above expression for each time t in a closed form without conducting any iterations.

How about initial weights? Inside θ, the weights of the DN are initialized randomly at $t = 0$. There are $k+1$ initial neurons in the Y area, and $V = \{\dot{\mathbf{v}}_i \mid i = 1, 2, ..., k+1\}$ is the current synaptic vectors in Y. Whenever the network takes an input $\mathbf{p}$, compute the pre-responses in Y. If the top-1 winner in Y has a pre-response lower than almost perfect match $m(t)$, activate a free neuron to fire. Eq. (9) showed that the initial weights of this free neuron is multiplied by a zero and therefore do not affect its updated weights.

Weng [11] proved that DN-1 computes the ML-estimator of all observations from the sensory space X and motor space Z using a large constant time complexity for each time t. Although DN learns incrementally, such a DN is error-free for learning any complex Turing machines, including any universal Turing machines.

VI. DN Experiments

The recent experimental results [15], [16] of DN work here include (1) vision that includes simultaneous recognition and detection and vision-guided navigation on MSU campus walkways, (2) audition to learn phonemes with a simulated cochlea and the corresponding behaviors, (3) acquisition of English and French in an interactive bilingual environment as well as its the corresponding behaviors, and (4) exploration in a simulated maze environment with autonomous learning for vision, path cost, planning, and selection of the least-cost plan. Due to the maximum likelihood optimality, there are no local minima problems for these highly nonlinear systems and the performance data were impressive [15], [16].

GENISAMA LLC, a startup that the author created, has produced a series of real-time machine learning products, as human-wearable robots. They are the first products ever existed as APFGP robots for practitioners to produce various kinds of intelligent auto-programed software.

VII. Conclusions

Although public and media have gained an impression that deep learning has approached or even exceeded human level performance on certain tasks (such as object recognition from static images), this paper has raised PSUTS which seems to question such claims. The author hopes that the exposure of PSUTS is beneficial to AI credibility and the future healthy development of AI, especially with the concepts of developmental errors and the framework of ML-optimal lifetime learning. In a model sense, each brain is probably ML-optimal if we consider the DN model here, but what affects human brains and developmental learning machines includes other factors such as the bodies, learning experiences and computational resources.

References

[1] Weng, J. *et al.* Autonomous mental development by robots and animals. *Science* **291**, 599–600 (2001).

[2] Weng, J. *Natural and Artificial Intelligence: Introduction to Computational Brain-Mind* (BMI Press, Okemos, Michigan, 2019), second edn.

[3] Weng, J. Autonomous programming for general purposes: Theory. *International Journal of Huamnoid Robotics* **17**, 1–36 (2020).

[4] Weng, J. Conscious intelligence requires developmental autonomous programming for general purposes. In *Proc. IEEE International Conference on Development and Learning and Epigenetic Robotics*, 1–7 (Valparaiso, Chile, 2020).

[5] Graves, A. *et al.* Hybrid computing using a neural network with dynamic external memory. *Nature* **538**, 471–476 (2016).

[6] Raayoni, G. *et al.* Generating conjectures on fundamental constants with the ramanujan machine. *Nature* **590**, 1–176 (2021).

[7] Weng, J. On post selections using test sets (PSUTS) in AI. In *Proc. International Joint Conference on Neural Networks*, 1–8 (Shengzhen, China, 2021).

[8] Russakovsky, O. *et al.* ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* **115**, 211–252 (2015).

[9] Weng, J., Ahuja, N. & Huang, T. S. Learning recognition and segmentation using the Cresceptron. *International Journal of Computer Vision* **25**, 109–143 (1997).

[10] Weng, J. Why have we passed "neural networks do not abstract well"? *Natural Intelligence: the INNS Magazine* **1**, 13–22 (2011).

[11] Weng, J. Brain as an emergent finite automaton: A theory and three theorems. *International Journal of Intelligent Science* **5**, 112–131 (2015).

[12] Weng, J., Zheng, Z. & Wu, X. Developmental Network Two, its optimality, and emergent Turing machines. U.S. Provisional Patent Application Serial Number: 62/624,898 (2018). Published.

[13] Weng, J. & Luciw, M. Dually optimal neuronal layers: Lobe component analysis. *IEEE Trans. Autonomous Mental Development* **1**, 68–85 (2009).

[14] Minsky, M. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine* **12**, 34–51 (1991).

[15] Zheng, Z., Wu, X. & Weng, J. Emergent neural Turing machine and its visual navigation. *Neural Networks* **110**, 116–130 (2019).

[16] Weng, J., Zheng, Z., Wu, X. & Castro-Garcia, J. Auto-programming for general purposes: Theory and experiments. In *Proc. International Joint Conference on Neural Networks*, 1–8 (Glasgow, UK, 2020).