

PUBLIC SUBMISSION

As of: 9/16/21 10:53 AM
Received: September 15, 2021
Status: Pending_Post
Tracking No. ktl-10xi-7thj
Comments Due: September 15, 2021
Submission Type: Web

Docket: NIST-2021-0004

Artificial Intelligence Risk Management Framework

Comment On: NIST-2021-0004-0001

Artificial Intelligence Risk Management Framework

Document: NIST-2021-0004-DRAFT-0082

Comment on FR Doc # 2021-16176

Submitter Information

Name: Juyang Weng

Address:

Okemos, MI, 48864

Email:

Phone:

General Comment

As an AI expert who initiated a direction called Autonomous Mental Development (AMD), I inform the US Government with the following three risks that the AI field is facing and two AI bottlenecks:

Risk 1: Shallow fitting of Data. The ImageNet Competition 2012 is laudable to have attracted investments from companies like Google into neural networks, like Convolutional Neural Networks (CNN) and Long Short Time Memory (LSTM), trained by a gradient-based method such as error backprop for a shallow fitting of “big data”. Such methods are both shallow and brute-force because data fitting is easy if a network has a huge number of parameters (see the nearest neighbor classifier in Att. 1). They do not generalize well if the networks are correctly tested (See Risk 2).

<http://www.cse.msu.edu/~weng/research/2021-06-28-Report-to-Nature-specific-PSUTS.pdf>

Risk 2: Protocol flaws. There is a rarely reported stage in report generation called Post-Selections: After many networks have been trained only report the luckiest network. Specifically, Post-Selections are technically flawed protocols because of one or more of the following flaws: (1) a lack of cross-validation for the hand-picked partition between the training set and test set, (2) a flawed use of test sets (see, e.g., PubPeer posts about LeCun et al. Nature 2015, Graves et al. Nature 2016, Li Fei-Fei et al. PAMI 2006, and Fig. 1 of Att. 1), (3) a lack of verification of the random seed luck for initial weights of a network, (4) a lack of verification of the hyper parameter luck of a network architecture, (5) the Post-Selection stage is typically not reported, (6) not reporting many less lucky networks, (7) a use of much human labor to hand-tune architecture parameters to fit a static “big data” set, (8) Some AI Competitions (like ImageNet) were improperly managed thus leaving time for “cracking” test sets and not caring how much human interventions, computational resources and manpower each team used.

Risk 3: “Big data” flaws. Pre-collected “big data” are wrong (e.g., violation of sensorimotor recurrence,

see Theorem 1 in Att. 1) and nonscalable (a lack of invariant rules, see Theorem 2 and observations 1 and 2 in Att. 1). Such machines do not know that they made mistakes nor how to recover from these mistakes. (E.g., for lane keeping, the human driver must periodically apply force to the steering wheel since the lane-keeping machines do not know they miss a plane and how to move back.) Thus, such machines are not trustable for even simple tasks like lane keeping. They still need a human to watch closely and to correct errors. Conscious learning is necessary to deal with these trustability and scalability issues, but conscious learning has not been published until Weng 2020 (See Att. 4) and could be overlooked for 20 more years like AI long overlooked the first Convolutional Networks (CNNs) for 3D (Cresceptron). Conscious learning is necessary to be trustable and scalable.

Bottleneck 1: Lack of Education for Conscious Learning. Universities have updated their curricula considerably, but they are still ill-suited for the required breadth and depth of Consciously Learning. The material in Weng, Natural and Artificial Intelligence (BMI Press) is required for AI experts to catch up with conscious learning. This is the first AI bottleneck.

Bottleneck 2: Lack of brain-scale real-time learning chips. Although brains learn in real time because of the sensorimotor recurrence requirement (see Risk 3), this necessary condition for trustable intelligence has been overlooked. Real-time learning requires brain-scale neural network chips, but chip manufacturers are not aware of this market. They do not know great pay-offs will come from a brain-scale real-time learning Developmental Network (DN). This is the 2nd AI bottleneck.

Attachments:

Att. 1: PostSelection-ArXiv-web.pdf (an archival journal manuscript of Att. 2 and 3, also from arXiv).

Att. 2: PSUTS-IJCNN2021rvsd-cite.pdf

Att. 3: Post-Selection-ICDL-2021rvsd-cite.pdf

Att. 4: ConsciousL-ICDL-2020rvsd-cite.pdf

Attachments

Att1-NNsubm-2021-09-09-web

Att2-PSUTS-IJCNN2021rvsd-cite

Att3-PostSelection-ICDL-2021rvsd-cite

Att4-ConsciousL-ICDL-2020rvsd-cite

Neural Networks

Post-Selections in AI and How to Avoid Them

--Manuscript Draft--

Manuscript Number:	
Article Type:	Article
Section/Category:	Mathematical and Computational Analysis
Keywords:	Experimental Protocol; Error-Backprop; Deep Learning; Performance Evaluation; Maximum Likelihood; Turing Machines
Corresponding Author:	Juyang Weng, Ph.D. Michigan State University East Lansing, Michigan UNITED STATES
First Author:	Juyang Weng, Ph.D.
Order of Authors:	Juyang Weng, Ph.D.
Abstract:	Neural network based Artificial Intelligence (AI) has reported increasing scales in experiments. However, this paper raises a rarely reported stage in such experiments called Post-Selection alter the reader to several possible protocol flaws that may result in misleading results. All AI methods fall into two broad schools, connectionist and symbolic. The Post-Selection fall into two kinds, Post-Selection Using Validation Sets (PSUVS) and Post-Selection Using Test Sets (PSUTS). Each kind has two types of post-selectors, machines and humans. The connectionist school received criticisms for its "black box" and now the Post-Selection; but the seemingly "clean" symbolic school seems more brittle because of its human PSUTS. This paper first presents a controversial view: all static "big data" are non-scalable. We then analyze why error-backprop from randomly initialized weights suffers from severe local minima, why PSUVS lacks cross-validation, why PSUTS violates well-established protocols, and why every paper involved should transparently report the Post-Selection stage. To avoid future pitfalls in AI competitions, this paper proposes a new AI metrics, called developmental errors for all networks trained, under Three Learning Conditions: (1) an incremental learning architecture (due to a "big data" flaw), (2) a training experience and (3) a limited amount of computational resources. Developmental Networks avoid Post-Selections because they automatically discover context-rules on the fly by generating emergent Turing machines (not black boxes) that are optimal in the sense of maximum-likelihood across lifetime, conditioned on the Three Learning Conditions.
Suggested Reviewers:	

Post-Selections in AI and How to Avoid Them

Juyang Weng

Department of Computer Science and Engineering

Cognitive Science Program

Neuroscience Program

Michigan State University, East Lansing, MI, 48824 USA

and

GENISAMA LLC

4460 Alderwood Drive, Okemos, Michigan 48864 USA

Post-Selections in AI and How to Avoid Them

Juyang Weng^{1,2,3,4}

¹Department of Computer Science and Engineering

²Cognitive Science Program

³Neuroscience Program

Michigan State University, East Lansing, MI, 48824 USA

⁴GENISAMA LLC, 4460 Alderwood Drive, Okemos, Michigan 48864 USA

Abstract

Neural network based Artificial Intelligence (AI) has reported increasing scales in experiments. However, this paper raises a rarely reported stage in such experiments called Post-Selection alter the reader to several possible protocol flaws that may result in misleading results. All AI methods fall into two broad schools, connectionist and symbolic. The Post-Selection fall into two kinds, Post-Selection Using Validation Sets (PSUVS) and Post-Selection Using Test Sets (PSUTS). Each kind has two types of post-selectors, machines and humans. The connectionist school received criticisms for its “black box” and now the Post-Selection; but the seemingly “clean” symbolic school seems more brittle because of its human PSUTS. This paper first presents a controversial view: all static “big data” are non-scalable. We then analyze why error-backprop from randomly initialized weights suffers from severe local minima, why PSUVS lacks cross-validation, why PSUTS violates well-established protocols, and why every paper involved should transparently report the Post-Selection stage. To avoid future pitfalls in AI competitions, this paper proposes a new AI metrics, called developmental errors for all networks trained, under Three Learning Conditions: (1) an incremental learning architecture (due to a “big data” flaw), (2) a training experience and (3) a limited amount of computational resources. Developmental Networks avoid Post-Selections because they automatically discover context-rules on the fly by generating emergent Turing machines (not black boxes) that are optimal in the sense of maximum-likelihood across lifetime, conditioned on the Three Learning Conditions.

Keywords: Experimental Protocols, Error-Backprop, Deep Learning, Performance Evaluation, Maximum Likelihood, Turing Machines

1. Introduction

AI research dates back at least to early 1910 when Leonardo Torres y Quevedo built a chess end game player called El Ajedrecista [1]. In 1950, Alan Turing published his now celebrated paper [2] titled *Computing Machinery and Intelligence*. Turing [2] was 5 impressive to have discussed a wide variety of considerations for machine intelligence, as many as nine categories. Unfortunately, he suggested to consider what is now called the Turing Test that has inspired and misled many AI researchers.

Much progress has been made in AI since then and many methods have been developed to deal with AI problems. As the scope of this paper, we will focus on generalization. All AI methods fall into two schools [3], symbolic and connectionist, although 10 many published methods are a mixture of both.

1.1. Symbolic school

Symbols are used in many AI methods (e.g., states in HMMs, nodes in Graphical Models and attributes in SLAM). Although symbols are intuitive to a human 15 programmer since he defines the associated meanings, symbols are static and have some fundamental limitations that have not received sufficient attention.

The symbolic school [4] assumes a micro-world in 4D space-time in which a set of objects or concepts, e.g., $L = \{l_1, l_2, \dots, l_n\}$, is assumed to be uniquely defined among many human programmers and their computers, represented by a series of symbols in 20 time $\{l_1(t), l_2(t), \dots, l_n(t) | t_0 \leq t < t_1\}$. The correspondences among all these symbols $\{l_i\}$ of the same object across different times are known as “the frame problem” [4] in AI which means that the programmer must manually link every symbol along time with its corresponding physical object. In computer vision, the symbolic school assumes a single symbol o_i , for all its 3D positions in its 3D trajectory $\{\mathbf{x}(t) | t_0 \leq t \leq t_1\}$ 25 and uses certain techniques, such as feature tracking through video (e.g., for driverless

cars). Therefore, the symbolic school is based on human-handcrafted set of symbols and their assumed meanings. Marvin Minsky wrote that symbols are “neat” [5], but in fact, symbols are “neat” mainly in a single human programmer’s understanding but not between different programmers and not in relating computer programs to a real world.

30 We will see the Developmental Network (DN) model of a brain is free from any symbols in its full version. Abstract symbols correspond to action/state vectors in the motor area of DN. Therefore, the frame problem is automatically solved through emergent action/state vectors in a physically grounded DN, without using any symbols in the DN’s internal representations.

35 A major problem for symbolic AI is the generalization issue of symbols as defined here.

Definition 1 (Brittleness of static symbols). *Suppose a symbolic AI machine $M(L)$ designed for a handcrafted set L of symbols is applied to a real world that requires a new set of symbols L' , with $L \cap L' \neq \emptyset$, $M(L)$ fails without a human programmer who 40 handcrafts an appropriate mapping function $f : L' \mapsto L$ that maps every element of L' to an element in L so that $M(f(L'))$ works correctly as before.*

Many expert systems (e.g., CYC, WordNet and EDR [6]) and “big data” projects [7] require a human programmer to be in the loop of handcrafting such a mapping f during deployment. For example, a machine M developed in Florida is deployed in Michigan 45 but Michigan has snow but Florida does not. Because it is extremely challenging for a human programmer to understand many implicit limitations of $M(L)$, the mapping f that the human handcrafts typically makes $M(f(L'))$ fail, resulting in the well-known high brittleness of symbolic systems.

Due to emergent representations as numeric vectors, a DN robot discussed below 50 learns snow settings and the snow concept when it sees snow scenes for the first time, because there are no symbols that correspond to snow in DN’s representations.

In general, the developmental methods to be discussed below automatically address such new concept problems without a need for a human programmer to be in the loop of handcrafting a symbolic mapping f during a deployment. In this paper, the author 55 will further argue that the symbolic school suffers from human PSUTS.

1.2. Connectionist School

The connectionist school claimed to be less brittle [8, 9]. However, a network is egocentric—meaning that the agent starts from its own (neural) network, instead of a symbolic world. It must learn from the external *world* without a handcrafted, 60 *world-centered* object model. Although connectionist methods often assume some task-specific symbols, e.g., a static set L of object labels, they also assume a restricted world implicitly. Therefore, a connectionist model typically needs to sense and learn from a restricted world using a network. The use of L by any neural networks (e.g., ImageNet [10] and many other competitions) as a set of object labels is a fundamental 65 limitation that also causes the resulting system to be brittle for the same reason as the symbolic school.

Typically, a neural network is meant to establish only a mapping f from the space of input X to the space of class labels L ,

$$f : X \mapsto L \tag{1}$$

[11, 12]. X many contains a few time frames. Many video analysis problems, speech recognition problems, and computer game-play problems are also converted into this static input space so that the input space also includes L , so as to learn

$$f : X \times L \mapsto L. \tag{2}$$

Without a pressure of performance characterization during learning other than the performance of the final network, a self-organization map (e.g., SOM) has been used often as an unsupervised but slow learning method [13, 14, 15].

70 In contrast, with a pressure of performance characterization during learning, Cresceptron [16] used a “skull-closed” incremental-learning Hebbian-like scheme with receptive-field based competitions.

Other than the Hebbian mechanisms which are strictly “unsupervised” used by he Cresceptron and the DN explained below, two other types of learning schemes have 75 been published:

A Human handpicking features: after knowing the test set, humans handpick features,

reported explicitly [17, 18, 19] or implicitly as “weakly supervised” [19]. This author called them “skull-open” [20].

B Error-backprop: Locally train multiple networks each from a different set of random
80 weights. After the training, post-select the luckiest network. Report the luckiest network only [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38] but many publications do not report the post-selection stage at al, with few exceptions [27].

Below, this author will argue that (A) suffers from human Post-Selections and (B)
85 suffers from machine Post-Selections. While Cresceptron, the first deep learning for a 3D world, generates a single large network, it showed an impressive generalization power due to the use of the nearest neighbor scheme at every layer of an automatically generated deep network. This author will argue that Post-Selections in (A) and (B) suffer from weak generalizations (due to three types of lucks to be discussed below) and
90 did not count the cost of training multiple networks many of which were not reported.

By the way, *genetic algorithms* offer another approach to such network learning. These algorithms study changes in genomes across different life generations. However, many genetic algorithms do not deal with lifetime development [39, 40]. We argue that handcrafting functions of a genome as a Developmental Program (DP) seems to be a clean and tractable problem, which avoids the extremely high, cost of evolution on DP.
95 Many genetic algorithms further suffer from the PSUTS problems, since they often use test sets as training sets (i.e., vanished tests) as explained below.

Marvin Minsky [5] among many scholars complained that neural networks are “scruffy” or “black boxes”. This problem is not addressed holistically until the frame-
100 work of Emergent Turing Machine was introduced [41] into Developmental Networks (DNs) by the Developmental School discussed below. A lack of Emergent Turing Machine mechanisms or being “scruffy” in sample fitting appears to be the main cause of PSUTS in traditional neural networks trained by human feature-handpicking or error-backprop methods.

105 *1.3. Developmental School*

The main thrust of the Developmental School, formally presented 2001 by Weng and six co-authors [39] is the task-nonspecificity for lifetime development, known as Developmental Programs (DPs) that simulate the functions of genome without simulating the genome encoding. Although a DP generates a neural network, a DP is very different from a conventional neural network in the evaluation of performance across each life—all errors from the inception time 0 of each life is recorded and reported up to each frame time $t > 0$, as explained further below.

The first developmental program seems to be the Cresceptron by Weng et al. [42, 43, 16] which appears to be, as far as the author is aware, the first deep-learning Convolutional Neural Network (CNN) for a 3D world. As explained in [44, 45] other well-known CNNs for 3D recognition, although they do not use a generative DP, followed many key ideas of Cresceptron. Cresceptron seems to be the first incremental neural network whose evaluation of performance is across its entire “life” and only one network was generated (developed) from the given training data set.

Cresceptron did not deal with time. A developmental approach that deals with both space and time in a unfired fashion using a neural network started from Developmental Networks (DNs) [46] whose experimental embodiments range from Where-What Networks 1 (WWN-1) [47] to WWN-9 [48]. The DNs went beyond vision problems to attack general AI problems including vision, audition, and natural language acquisition as emergent Turing machines [41].

DNs overcame the limitations of the framewise mapping in Eq. (2) by dealing with lifetime mapping:

$$f : X(t-1) \times Z(t-1) \mapsto Z(t), t = 1, 2, \dots \quad (3)$$

where $X(t)$ and $Z(t)$ are the sensory input space and motor space, respectively, and \times denotes the Cartesian product of sets. A fundamental difference between Eq. (2) and Eq. (3) is that in the latter the Z space contains exclusively emergent vectors, instead of any symbols, so that the actions/states are incrementally taught and learned across a lifetime.

As we will see in Section 4.1, all the errors occurred during any time of each life is recorded and taken into account in the performance evaluation.

It is important to extend Eq. (3) to include the hidden area Y that generates international (hidden) representations. To model how Y -to- Y connections enable something similar to higher and dynamic order in Markov models (but not symbolic), the above lifetime mapping is extended to:

$$f : X(t-1) \times Y(t-1) \times Z(t-1) \mapsto Y(t) \times Z(t), t = 1, 2, \dots \quad (4)$$

Note that $Z(t-1)$ here is extremely important since it corresponds to the state of a Turing machine. Namely, all the errors occurred during any time of each life is recorded and taken into account in the performance evaluation.

Different from the static symbols in the symbolic school and the space of class labels L of static symbols in Eq. (2) of the connectionist school, the space $Z(t)$ of numeric vectors of the developmental school is free from symbols. Therefore, these states/actions are directly teachable or self-generative, inspired by brains [49, 50, 51, 52, 53, 51, 54]. This new symbol-free formulation is necessary to model not only brain's spatial processing [55] and temporal processing [56], but also Autonomous Programming for General Purposes (APFGP) [57]. Based on the APFGP capability, we open the door towards the next step—conscious learning [58]—learning while being partially and increasingly conscious. By *conscious learning*, we do not mean “open-skulledly” handcrafting general-purpose consciousness, which is probably too complicated to handcraft. But instead we enable fully autonomous machine learning while machines being partially conscious—autonomously learn more sophisticated consciousness skills using their partial, earlier, and simpler conscious skills across the lifetime.

This is the journal archival version of the earlier conference papers [59, 60] with significantly refined additional material and analysis.

In the following, we will discuss Post-Selection in Section 2. Section 3 addresses why error-backprop algorithms suffer from severe local minima problems. Section 4 explains how a Developmental Network solves the local minima problems, since only one network is needed for each life and the evaluation of performance across the entire

life. Section 5 discusses experiments. Section 6 provides concluding remarks.

2. Post-Selections

AI has made impressive progress, gained much visibility, and attracted the attention of many government officials. However, there are protocol flaws that have resulted in
160 misleading results.

First, let us consider three learning conditions that any fair comparisons of AI methods should take into account.

2.1. The Three Learning Conditions

Many AI methods were evaluated without considering how much computational
165 recourses are necessary for the development of a reported system. Thus, comparisons about the performance of the system have been tilted toward competitions about how much resources a group has at its disposal, regardless how many networks have been trained and discarded, and how much time the training takes.

Here we explicitly define the Three Learning Conditions for development of an AI
170 system:

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

The competing standard of the ImageNet competitions [61] did not include any of these three conditions. The AIML Contests [62] considered all the three in performance evaluation. In the following Subsection, we discuss why task-nonspecificity and incremental mode should be considered in any comparisons.

180 2.2. Task-specific vs. Task-nonspecific

A task-specific learning approach learns less because much is handcrafted by a human according to the given task. Furthermore, a task-specific method is brittle.

In Condition (1) of the Three Learning Conditions, the task-nonspecific learning paradigm is significantly different from the task-specific traditional AI paradigm as explained in Weng et al. 2001 [39]. In a task-specific paradigm, the system developer is given a task e.g., constructing a driverless car. Then, it is a human programmer who chooses a world model, such as a model of lane edges. Next, he picks an algorithm based on this world model, e.g., the well-known Hough transform algorithm [63, 64] for line detection which makes every pixel that is detected as edge cast votes for lines of all possible orientations o and distances d from the origin that go through the pixel. Then the top-two “peaks” of line parameters (o, d) that have received the highest votes are adopted to declare a line detected from the image. Here “edges” and “lane lines” are two symbolic concepts picked up by the programmer. Such systems will fail when lanes are unclear or totally disappear due to weather or road conditions, leading to a brittle system. Human brains appear to be more resilient.

In contrast, a task-nonspecific approach [39] not only avoids any symbolic model, but also does not require that a task is given. The desirable actions at any time are taught, tried, and recalled automatically by the learner based on system’s learned context q [65] that includes automatically figured-out goal and state, as well as the current input. The mapping function $f(\mathbf{z}, \mathbf{x}) = \mathbf{z}'$, representing the symbolic mapping $f_s(q, \sigma) = q'$, corresponds to a finite automaton. Weng has proven that the control of any Turing machine is a finite automaton [41]. Thus, this framework is of general-purposes in the sense of universal Turing machines. Any universal Turing machine is of general purposes, because it can read any program written for any purposes and run it for the purposes. Any neural network that learns a universal Turing machine becomes of general purposes in the sense of any programs, not just in the sense of any mappings like that in Eq. (1). Thanks to the absence of any world model, such as lanes, this task-nonspecific approach has a potential to be more robust than a world-model based approach. The task-nonspecific approach typically uses a neural network to learn because the need to learn vector mapping function $f(\mathbf{z}, \mathbf{x}) = \mathbf{z}'$. We will discuss internal response vector \mathbf{y} in Section 4 but task-nonspecificity holds true without \mathbf{y} .

2.3. Batch vs. Incremental Learning Modes

Neural network learning for the mapping f has two learning modes, batch learning and incremental learning.

215 With batch learning, a human first collects a set D of data (e.g., images) and then labels each datum with a desirable output (e.g., command of navigation or class label). A neural network is trained to approximate a mapping f in Eq. (1) or Eq. (2). Many batch-learning projects use an error-backprop method [23, 66, 24] which uses a gradient-based method to find a local minimum in error.

220 As we will discuss in Section 3, the gradient in the error-backprop method does not contain key information of many other data if the learning mode is incremental. Thus, error-backprop on a large data set does poorly using a purely incremental learning mode. Many used a block-incremental learning mode which suffers from the big data flaw in Theorem 1 below.

225 In contrast, all developmental methods cited here use incremental learning mode for long lifetimes, using a closed-form solution to the global lifetime optimization. The competition among neurons guarantees that the winner is the most appropriate neuron whose memory corresponds to the current working memory [67].

However, the batch and incremental learning modes are not capability-equivalent
230 [67]. The former requires all sensory inputs are available at a batch, independent of the corresponding actions. Therefore, the former is easier and also incorrect according to sensorimotor recurrence. By sensorimotor recurrence, we mean that sensory inputs and motor outputs are mutually dependent on each other in such a recurrent way that off-line collection of inputs are technically flawed. We have the following theorem:

235 **Theorem 1 (Big Data Flaw).** *All static “big data” sets used by machine learning violate the sensorimotor recurrence property of the real world.*

Proof: A learning agent at time $t - 1$, as shown in Eqs. (3) and (4) does not have the next sensory input from $X(t)$ available before the corresponding actions in $Z(t - 1)$ are generated and output, since the sensory input in $X(t)$ varies according to the agent actions in $Z(t - 1)$. As an example, turning head left or right will result in a different

image sensed. Therefore, all static “big data” sets violate the sensorimotor recurrence.

■

One may say that classifications of static images are fine. We do not agree, because even when a human (or machine) is looking at a static image, he uses attention
245 (e.g., context-based saccades) which is a sequence of actions. Each saccade results in a different fovea image.

Therefore, incremental learning is necessary for the *sensorimotor recurrence*. All batch-training methods use a static set of training data and, therefore, are inappropriate for any of them to claim near-human performance since the two learning problems are
250 different. This leads to the following theorem.

Consider a hierarchy of levels of object types, such as nails, fingers, palms, hands, arms, limbs, torsos, human bodies, etc. Because vision requires a high level l to understand natural scenes with abstraction of parts with invariances (e.g., all fingers of different scales, looks, and at different locations), each child needs an open-ended world
255 to learn to learn rules (e.g., finger-parts and hand-whole) instead of simple-minded pattern recognition of sensory images.

Theorem 2 (Nonscalability of Big Data without abstraction). *All static “big data” sets used by machine learning are nonscalable if they are treated as pattern recognition without rule abstractions.*

260 *Proof:* Suppose that a static data set D has shown the presence of k feature types defined at level 1 (e.g., edge pixels are a type). Suppose a combination of $k > 1$ feature types to level $l + 1$ type (e.g., straight line type is from multiple edge pixels) is defined from k types of feature types at level l , $l = 1, 2, \dots$. The number of samples for a l -level feature type requires at least k^l observations to discover all necessary within-type
265 equivalence (e.g., logic OR is at $l = 2$ with $k = 2$ logic features at $l = 1$, thus without rule abstraction (e.g., parts and whole), it requires $k^l = 2^2 = 4$ observations, corresponding to 4 rows in the truth table of logic OR). Since $f(l) = k^l$ is an exponential function in l , k^l quickly exceeds any fixed number of observations in the static data set D .

270 Rule abstractions deal with invariances. For example, a “what” concept is “where”-invariant and a “where” concept is “what”-invariant, as explained in [55, 68].

Section 4 discusses an optimal framework through which such abstractions can take place from learning simple rules during early life that enable learning of more complex rules during later life—called scaffolding [69].

275 Theorem 2 leads to two observations on **data fitting on a static data set**:

Observation 1: Any **data fitting on a static data set** without learning invariant concepts are nonscalable, including the n -fold cross-validation discussed below. Unfortunately, **data fitting on a static data set** is a norm in all ImageNet Contests [66]. Namely, the remaining subsections in this section analyze approaches that are nonscalable. For example, computer vision is not a “one-shot” pattern classification problem as argued by Li Fei-Fei et al. [19] (which was questioned in PubMed without responses), but rather a spatiotemporal problem to learn various invariant concepts present in cluttered natural scenes through autonomous attention saccades, as explained further in Observation 2.

285 **Observation 2:** Learning invariant concepts seem nonscalable for any **data fitting on a static data set** either, because there are too many images to be labeled by hand (e.g., all pixel locations) [55, 68]. Like a human baby, any scalable machine learning methods must be conscious through which the machine learner must consciously guess concepts (i.e., not just active learning [70]) (e.g., an object type) and verify their invariance rules (e.g., the where-invariance of a what concept). The state-based transfer in Theorem 8 of [56] explains how each concept state reduces the number of samples to be learned from an exponential k^l down to only kl (see Fig. 6 of [56] for intuition where $k = 10$ and $l = 3$). Thus, Section 4 not only addresses the non-scalability problems in this section, but is also necessary for conscious learning whose theory was 290 recently published in Weng 2020 [58] with some single-sensory-modality experimental results, but animal-level conscious robots that are multi-sensory and multi-motor have not yet been demonstrated. The availability of real-time learning brain-chip is a current bottleneck.

2.4. Fitting, validation and test errors

Given an available data set D , D is divided by a partition $P = (T, V, T')$ into three mutually disjoint sets, a training set T , a validation set V , and a test set T' so that

$$D = T \cup V \cup T'. \quad (5)$$

- 300 Two sets are disjoint if they do not share any elements. The validation set is possessed by the trainer, the test set should not be possessed by the trainer since the test should be conducted by an independent agency. Otherwise, V and T' become equivalent.

As we will see in Section 3, given any architecture parameter vector \mathbf{a}_i , it is unlikely that a single network initialized by a set of random weight vectors can result in an acceptable error rate on the training set, called fitting error, that the error-backprop training intends to minimize locally. That is how the multiple sets of random weight vectors come in. For k architecture vectors \mathbf{a}_i , $i = 1, 2, \dots, k$ and n sets of random initial weight vectors \mathbf{w}_j , the error back-prop training results in kn networks

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

Error-backprop locally and numerically minimizes the fitting error $f_{i,j}$ on the training set T .

- 305 [27] seems to have mentioned $n = 20$. [71] did not give n but seems to have mentioned 60 million parameters which probably means each \mathbf{w}_i and each \mathbf{a}_j combined to be of 60 million dimensional. Using the above example of $k = 3^{10} = 59049$, $kn \approx 1M$ networks must be trained, a huge number that requires a lot of computational resources to do number crunching and a lot of manpower to manually tune the range of hyper-parameters.

- 310 **Definition 3 (Distribution of fitting, validation and test errors).** *The distributions of all kn trained networks' fitting errors $\{f_{ij}\}$, validation errors $\{e_{ij}\}$, and test errors $\{e'_{ij}\}$, $i = 1, 2, \dots, k$, $j = 1, 2, \dots, n$ are random distributions depending on a specific data set D and its partition $P = (T, V, T')$. The difference between a validation error and a test error is that the former is computed from the same group using a group-possessed validation set V but the latter is computed by an independent agency using a group-unknown test set T' .*

We define a simple system that is easy to understand for our discussion to follow.

Definition 4 (Nearest neighbor classifiers with a confidence threshold). *Define a network stores the entire training set T . Suppose the input x matches the nearest sample s in T . If the distance between x and s is not larger than a confidence threshold d (a hyper-parameter), then the network outputs the associated label of the nearest sample s. Otherwise, the system outputs “unknown”.*

Namely, this system uses a lot of resources for over-fitting. It gives up if the distance is larger than d , but has a perfect fitting error (zero) for any positive d .

A neural network architecture has a set of hyper parameters represented by a vector \mathbf{a} , where each component corresponds a scalar 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, $A = \{\mathbf{a}_i | i = 1, 2, \dots, k\}$. Suppose there are 10 scalar parameters in each vector \mathbf{a}_i . For each scalar parameter x of the 10 hyper parameters, we need to validate the sensitivity of the system error to x . With uncertainty of x , we estimate its initial value as the mean \bar{x} , positively perturbed estimate $\bar{x} + \sigma_x$ (σ is the estimated standard deviation of x), and negatively perturbed estimate $\bar{x} - \sigma_x$. If each scalar hyper parameter has three values to try in this way, there are a total of $k = 3^{10} = 59049$ architecture parameter vectors to try, a very large number. For example, the initial threshold \bar{d} in the nearest neighbor classifier can be estimated by the average of nearest distance between a sample in V and the nearest neighbor in T and the σ_d be estimated by the standard deviation of these nearest distances.

Let us define the Post-Selection. Suppose that the trainer is first aware of the validation sets (or the test sets).

Definition 5 (Post selection). *A human programmer trains multiple systems using the training set T . 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 set V (or the test set T'). This is called Post-Selection—selection of one network from multiple trained and verified (or tested) networks.*

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

First, consider Post-Selection Using Validation Sets (PSUVS):

2.5. 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 luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches *the error of the luckiest-architecture and the luckiest initial weights from Post-Selection on Validation Sets*:

$$e_{i^*,j^*} = \min_{1 \leq i \leq k} \min_{1 \leq j \leq n} e_{i,j} \quad (6)$$

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

Similarly, a human PSUVS is a procedure wherein a human selects a system from multiple trained systems for $\{e_{i,j}\}$ using human visual inspection of internal representations of the system and their validation errors.

2.6. Cross-Validation

The above PSUVS is an absence of cross-validation [72]. Originally, the cross-validation is meant to mitigate an unfair luck in a partition of the dataset D into a training set T and a test set T' (empty validation set). For example, an unfair luck is such that every point in the test set T' is well surrounded by points in the training set T . But such a luck is hardly true in reality.

To reduce the bias of such a luck, an n -fold cross-validation protocol, $n \geq 2$, divides the data set D into n subsets of same size and conducts n experiments. The term “cross” refers to switching the roles of training and testing data. In the i -th experiment, the i -th subset is left out as the test set and the remaining $n - 1$ folds of data form the training set. Thus, the cross-validation protocol conducts n experiments, for $i = 1, \dots, n$, to obtain n errors, e_1, e_2, \dots, e_n . The *cross-validated error* is defined as the

average of errors from the n tests, to filter out the partition lucks:

$$\bar{e} = \frac{1}{n} \sum_{i=1}^n e_i \quad (7)$$

as well as the distribution of errors $\{e_j\}$.

365 The n different numbers here shows a distribution show a distribution $\{e_j\}$ to indicate how sensitive the error is to lucks, such as the number of partition pairs between a training set and a validation/test set, the number of tried random seeds for initializing network weights, the number of tried hyper-parameter vectors, or a combination thereof. The larger the n , the better the estimated standard deviation of $\{e_j\}$.

370 *2.7. Types of lucks in a Neural Network*

In a neural network, there are at least three kinds of lucks:

375 **Type-1 order lucks:** The luck in a partition P_i into a training set T_i and a test set T'_i from a data set D resulting in test error e_i , $i = 1, 2, \dots, n$. Different partitions correspond to different luck outcomes. This kind of outcome variation results in a variation of performance from different outcomes. Conventionally, this type of lucks is filtered out by cross-validation (e.g., n -fold cross-validation) as well as reporting the deviation of $\{e_i\}$ during the cross-validation. However, such cross-validation and deviation have hardly published for neural networks and reported. The smaller the average \bar{e} of $\{e_i\}$, the more accurate the trained network is; the smaller the standard deviation of $\{e_i\}$, the more trustable the average error \bar{e} is.

380 **Type-2 weights lucks:** As we will discuss below, weights specify the role assignment for all the neurons in the neural network. A random seed value determines the initialization of a pseudo-random number generator, which gives initial weights \mathbf{w}_i for a neural network $N(\mathbf{w}_i)$, resulting in a test error e_i , $i = 1, 2, \dots, n$, after training of these n networks and testing on T' . It is unknown that such a luck will be carried over to a new test set T'' that is outside the data set D but was drawn from the same distribution of S . Because a neural network might not capture the internal rules of the training set T , this paper argues that a statistical validation of the reported error should be performed by reporting the distribution of $\{e_i | i = 1, 2, \dots, n\}$, where e_i is from a different initial weight vector \mathbf{w}_i . For example, Krizhevsky et al. [71] reported 60 million

parameters, mostly in \mathbf{w}_i but only the luckiest e_i was reported. The smaller the average \bar{e} of $\{e_i\}$, the more accurate the trained network is; the smaller the standard deviation σ of $\{e_i\}$, the less sensitive the trained neural network is to the initial weights and thus the accuracy is more trustable for real applications. For i.i.d. (identically independently distributed) errors, we can expect that doubling the number n will reduce the expected variance of \bar{e} by a factor $1/\sqrt{2}$, since the expected variance of n random numbers is about σ^2/n .

Type-3 architecture lucks: The initial hyper-parameter vector \mathbf{a}_j of the neural network gives an error e_j , $j = 1, 2, \dots, k$. Because such a luck of \mathbf{a}_j might not capture the internal rules of the training set T_j , this paper argues that a statistical validation of the reported error estimate should be performed and the distribution of $\{e_j\}$ be reported. In our above example, the number of distinct hyper-parameter vectors to be tried is $k = 3^{10} = 59049$. The smaller the average \bar{e} of $\{e_j\}$, the more accurate the trained network is; the smaller the sample variance of $\{e_j\}$, the more trustable \bar{e} is, namely, the average error \bar{e} is less sensitive to the initial hyper-parameters of the network. For example, the threshold d of the nearest neighbor classifier in Definition 4 might result in a large deviation. A good way is to reduce the manual selection nature of such hyper-parameters. For example, all hyper-parameters are adaptively adjusted from the initial hyper-parameters that are further automatically computed from system resources, e.g., the resolution of a camera, the total number of available neurons, and the firing age of each neuron [67].

For notation clarity in the discussion that follows, index j is used in Type 3 to distinguish index i in type 2, but the above three types of lucks are all different.

Let us discuss the case of a developmental network, such as Cresceptron [16] and DN [41]. Type-1 cross-validation is not needed because of reporting of a lifetime error. In other words, errors of all new tests in each life are taken into account throughout the lifetime. Type-2 validation is not needed because all different random weights \mathbf{w}_i leads to the function-equivalent neural network under certain conditions. For example, in top- k competition, with $k = 1$ different \mathbf{w}_i give the exactly the same neural network and with $k > 1$ different \mathbf{w}_i give almost the same neural network. The distribution of lifetime errors $\{e_i\}$ is expected to have a negligible deviation across different initial

weight vectors \mathbf{w}_i , given the same Three Learning Conditions. Type-3 validation might be useful but is expected to be negligible since the most obvious parameters such as learning rate and momentum of learning rate is automatically and optimally determined by each neuron, not handcrafted, as in LCA [67]. The synaptic maintenance automatically adjusts all receptive fields [73, 74] so that the neural network performance is not sensitive to the initial hyper-parameters.

In contract, a batch-trained neural network typically uses a Post-Selection to pick the luckiest network without cross-validation for either of the above three types of lucks, e.g., in ImageNet Contest [61]. Namely, errors occurred during batch training of the network before the network is finalized and how long the training takes are not reported. Below, Fig. 5 will show a huge difference between the luckiest CNN with error-backprop and the optimal DN. Many researchers have claimed error-backprop works without providing much-needed three types of validations.

Next, let us discuss Types 2 and 3 validations which are new for neural networks but hardly done.

2.8. Post-Selection with Types 2 and 3 Average-Validations

Type-1 cross-validation should be nested inside the Types 2 and 3 validations, but this triple-nested protocol could be too computationally expensive. Below, we delay Type-1 cross-validation till after Type 2 and Type 3 validations.

Assume that we use n random weight vectors \mathbf{w}_i and k grid-search hyper parameters \mathbf{a}_j . Each combination of \mathbf{w}_i and \mathbf{a}_j gives an error $e_{i,j}$ from the corresponding validation set. To reduce the effect of such a luck for each vector \mathbf{w}_i , an average of $e_{i,j}$ over n values of i should be used instead of the minimum in Eq. (6). This leads to *the random-weights validated error for the luckiest architecture* from PSUVS:

$$\mathbf{a}^* = \arg \min_{1 \leq j \leq k} \frac{1}{n} \sum_{i=1}^n e_{i,j}. \quad (8)$$

We dropped the term “cross” because this validation examines other random seeds without switching the roles between training and testing.

Similarly, we define *the hyper-parameter validated luckiest initial weights* from

PSUVS:

$$\mathbf{w}^* = \arg \min_{1 \leq i \leq n} \frac{1}{k} \sum_{j=1}^k e_{i,j}. \quad (9)$$

We dropped the term “cross” for the same reason.

From a statistical point of view, the initial hyper parameter vector \mathbf{a}^* and the random initial weights \mathbf{w}^* validated above through averages should be more robust in real
445 applications than those without average-validation in Eq. (6).

For both the luckiest \mathbf{a}^* and \mathbf{w}^* , the standard deviation under min should be reported to show how sensitive the reported performance is to the validation process. If the variation is large, the corresponding network is not very trustable in practice.

450 We also need to be aware of another protocol flaw: Random seeds and hyper parameters are all coupled. Under such a coupling, Type 2 validation seems unnecessary with $n = 1$ but the search of the luckiest weights is embedded into the search for the luckiest hyper-parameter vector where each hyper parameter vector uses a different seed. Similarly, Type-3 validation seems unnecessary with $k = 1$ but the search of the
455 luckiest hyper-parameter vector is embedded into the search for the luckiest weights, where each random seed uses a different hyper parameter vector.

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 a validation set does not guarantee doing well on a test set. Typically, due to a very
460 large number of samples, availability of validation sets and unavailability of test sets in a properly managed contest, principles of Post-Selection should cause the validation error rate to be smaller than the test error rate. (However, in Table 2 of [71], the test error rate is smaller than the validation error for 7CNNs, causing a reasonable suspicion that PSUTS could be used instead of PSUVS.)

465 The following subsection discusses the luckiest network with the luckiest hyper-parameter vector \mathbf{a}^* and the luckiest initial weights \mathbf{w}^* .

2.9. The Luckiest Network from a Validation Set

Many people may ask: Are there any technical flaws in at least PSUVS, since it does not use the test sets? We analyze the luckiest network in this section and reach

⁴⁷⁰ a conclusion that any post-selection is technically flawed and results in misleading results, including both PSUVS and PSUTS. However, in general, Type-1 cross-validation is to filter out lucks in data partition that a typical user does not have during a deployment of the method. Namely, it is a severe technical and protocol flaw in reporting only the luckiest network, regardless the post-selection uses validation sets or test sets.

⁴⁷⁵ This conclusion has a great impact on evolutional methods that often report only the luckiest network, instead of those of all networks in a population. Namely, the performances of all individual networks in an evolutionary generation should be reported.

⁴⁸⁰ For simplicity, we assume that the space S , from which random samples in D are drawn, is static. Our conclusions here can be readily extended to a time varying D but the technical flaws are even worse.

⁴⁸⁵ From the sample space S , randomly draw a data set D . D is partitioned into three mutually disjoint sets, training set T , validation set V and test set T' , so that Eq. (5) holds true. For realistic applications, we should assume that T , V and T' are mutually independently drawn from S so that T , V and T' are mutually independent. Identically independently distributed (i.i.d.) is a sufficient condition, but we do not need such a restrictive condition because temporal-dependency often occurs in lifetime development. Namely, we only need that any three vectors from T , V and T' , respectively, are mutually independent.

Using the training set T , one trains kn networks, where k and n are the number of hyper-parameter vectors \mathbf{a} 's and random weight vectors \mathbf{w} 's, using a training algorithm (e.g., error-backprop),

$$N(\mathbf{a}_i, \mathbf{w}_j) \leftarrow f_{\mathbf{a}_i, \mathbf{w}_j}(T). \quad (10)$$

⁴⁹⁰ This is like a teacher trains kn students in a class. The teacher knows that the fitting error on T does not predict the validation error well, due to the possibility of over fitting. One extreme example is the above nearest-neighbor classifier with confidence $d = 0$.

The teacher then tests each $N(\mathbf{a}_i, \mathbf{w}_j)$ on the validation set V to get $e_{i,j}$. This is like the teacher observes the performance of kn networks in a mock exam.

⁴⁹⁵ The teacher then post-selects and reports only the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$

whose validation error e_{i^*,j^*} is minimum in Eq. (6). This is like the teacher colludes with the Educational Test Service (ETS) so that the ETS only reports the luckiest network but not all remaining $kn - 1$ networks to cover up.

2.10. Luckiest Network with Type-1 Cross-Validation

500 Suppose that a user has bought the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ and test on his new test data T' randomly drawn from S , independent of T and V . The luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ that reached the minimum error rate in V does not mean that it reaches the minimum error rate in T' . Because T' is independent of T and V , and $\mathbf{a}_{i^*}, \mathbf{w}_{j^*}$ are luckiest on a particular pair (T, V) only, we need to compute the expected 505 error rate of $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ on T' .

Theorem 3 (Type-1 cross-validation of the luckiest). *The luckiest network on validation set gives an error rate that is approximately the average error in Type-1 cross-validation, supposing that, in an n -fold cross validation, n folds of data are drawn i.i.d. (independently and identically distributed) among folds from data set D , but individual 510 samples inside each fold do not need to be i.i.d.*

Proof: Let F denote the event that both the training set and validation set are from a fixed data set D from S . Consider in a real application, n tests were conducted on the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ using T'_i , $i = 1, 2, \dots, n$, where each partition $P_i = (T_i, V_i, T'_i)$ in each of the i -th training and test pair is drawn from the real application space S . We compute the average error rate from the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$:

$$\begin{aligned} e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}) &= \frac{1}{n} \sum_{i=1}^n e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i) \\ &\approx \frac{1}{n} \sum_{i=1}^n e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i | F) \end{aligned} \quad (11)$$

where the term $e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i)$ means the error of the luckiest network using training set T_i , validation set V_i , and test set T'_i , and $e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i | F)$ means the same but T_i, V_i, T'_i are all from the same D as one does in n -fold cross-validation.

■

520 Note, the n -fold i.i.d. is weaker than i.i.d. for all samples. In practice, i.i.d. is rarely true even for pattern recognition problems, such as image classification due to sequential attention discuss above. Also note that the left side of \approx sign in Eq. (11) is expected larger because the data T_i, V_i, T'_i on the right side are all from a fixed D but the left side does not have such a restriction.

525 The above theorem tells us that the error rate of the luckiest network from a single validation set in PSUVS is misleading without any partition validation. This is because the error rate is a random function, depending on not only many random initial weights, many hyper parameters, and local lucks of error-backprop, but also a particular partition (T, V, T') . This seems especially true if the data D were made public and
530 overworked during 2010-2014 [61, p. 213].

In practice, when we report an error rate $e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ which is always a random number x , depending on how much hand tuning is done, how much computational resources are used for a large-scale search for the random seeds and hyper-parameters, as well as the validation or a lack thereof. We should also report the distribution of this
535 random number x , such as the maximum, 75%, 50%, 25%, and the minimum value of x , over multiple training-and-test pairs in cross-validation, random seeds and hyper-parameters. Otherwise, the error rate, if only as a single number x , is misleading, since users of this learning method or buyers of the luckiest network do not have the same partition luck.

540 Up to now, this author has not found any published papers that report not only the luckiest network from error-backprop but also Type-1, Type-2 and Type-3 validations. Many papers do not report the post-selection stage at all [24, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38], except [27], let alone whether the reported error is from the validation error V or the test set T' .

545 Next, we discuss Post-Selections Using Test Set (PSUTS). There are two kinds of PSUTS, machine PSUTS and human PSUTS.

2.11. Machine PSUTS

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

550 (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 luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches the minimum, called *the error of the luckiest architecture and the luckiest initial weights from Post-Selection on Test Set*:

$$e'_{i^*, j^*} = \min_{1 \leq i \leq k} \min_{1 \leq j \leq n} e'_{i,j}. \quad (12)$$

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. (7) to give *the error of the luckiest architecture with validated weights from PSUTS*:

$$\mathbf{a}_{*, j^*} = \arg \min_{1 \leq j \leq k} \frac{1}{n} \sum_{i=1}^n e'_{i,j}. \quad (13)$$

But the above error is still flawed since each term under minimization has peeked into test sets. Instead, it is better to use Eq. (8) which does not use the test sets. Of course, 555 the test error rate of that in Eq. (8) tends to be larger than that from Eq. (13).

A similar discussion can be made for *the error of the luckiest initial weights with validated architecture from PSUTS*. Do not peek 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 560 it test-vanished PSUTS.

In general, the more free parameters a network has, the more likely the network can report an artificially small error as in Eq. (12). That is why we need the computational resource in the Three Learning Conditions.

Although PSUVS has flaws of post-selection and a lack of three types of validation, 566 the key difference between PSUVS and PSUTS does not guarantee that PSUVS reports a low error rate as PSUTS. In fact, it is expected that the luckiest network from PSUVS does better on a validation set V than on a test set T' because the Post-Selection did not “see” the test set T' but “saw” the validation set V . Likewise, it is expected that the luckiest network from PSUTS does better on the test set T' than on a validation set

570 V because the Post-Selection did not “see” the validation set V but “saw” the test set T' . In the following paragraph, we discuss that this expectation is reversed in Table 2 of [71, page 88].

In ImageNet Contest 2012, the test sets were released to competition teams over 2.3 months ahead of the output-result submission date. Although the class labels were 575 not attached to the test sets other than being available indirectly through an online test server provided by the contest organizers, it was not difficult to “crack” a test set by manually hand-labeling the test set. The first author of [71] seems not sensitive to the fundamental difference between a validation set and a test set by writing: “in the remainder of this paragraph, we use validation and test error rates interchangeably”.
580 By “we cannot report test error rates for all the models that we tried” [71, page 88], there is no evidence to rule out what he meant was the possibly “cracked” test set is not necessarily exactly the same as the original test set. But in Table 2 of [71, page 88], the 7NNs did worse on the validation set (possessed) than the test set (if not “cracked” and searched for minimization like in Eq. (12)). This reversed our expectation in the previous paragraph. Is it an evidence of using PSUTS instead of PSUVS?

Another interesting phenomenon that is consistent with the likely use of PSUTS instead of PSUVS is that the SuperVision Team of ImageNet Contest 2021 did not submit any output results for “the fine-grained classification task, where algorithms would classify dog photographs into one of 120 dog breeds” [61, footnote, p214]. It 590 appears that cracking “120 dog breeds” is harder than cracking “a list of object categories present in the image” where the class labels are all available in the provided training sets. [71] lacks due transparency about the post-selection stage except that Geoffrey Hinton admitted the “luckiest” network in his brief PubPeer response to questions raised on PubPeer towards [24].

595 For more examples, see Fig. 1 from [75, Fig. 7], error-backprop consistently results lower validation accuracies than the test accuracies (about 0.5% lower compared to about 0.1% lower in [71]). Are they other evidences of using PSUTS instead of PSUVS, similar to [71]? The availability of test sets to the programmers in a project seems to be indeed addictive towards PSUTS, away from PSUVS. The standard deviation around 1% is clashes with our Theorem 4. Our experience with our own ex-
600

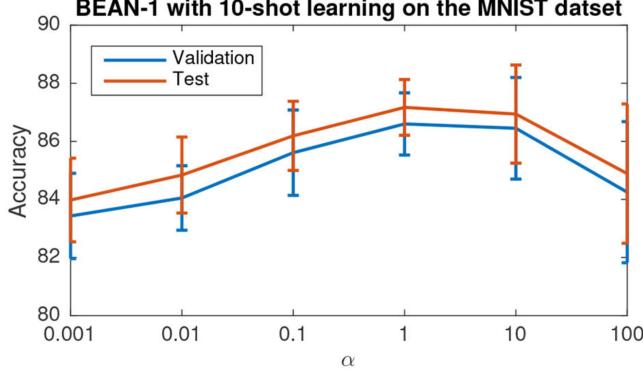


Figure 1: The average \bar{e} and the standard deviation of $\{e_i\}$ for different values of a regularization hyper parameter α . Adapted from [75]

periments with error-backprop training for CNN indicated that the maximum and the minimum values of the distribution of fitting accuracies are drastically different for different random seeds, with fitting accuracies spreading uniformly between 20% and 90%. Section 3 will discuss why. If Theorem 4 in Section 3 is correct, the deviation
605 bars seem too small and the 20 runs in Fig. 7 of [75] could be the best 20 among many more random-seeds the programmer has tried. We hope that authors provide the source program.

2.12. Implications of PSUTS

Although the set $\{e_{i,j} | i = 1, 2, \dots, n; j = 1, 2, \dots, k\}$ is large, it is necessary to
610 present some key statistical characteristics of its distribution. For example, rank all errors in decreasing order, for each type of errors, fitting, validation and test. Then give the maximum, 75% (in ranked population), 50% (median), 25%, the minimum value, and the standard deviation of these kn values for the fitting errors, validation errors. and test errors, respectively, not just the standard deviation in Fig. 1 . Such more
615 complete information of the distribution is critical for the research community to see whether error-backprop can indeed avoid local minima in deep learning as some authors claimed. Furthermore, such information is also important for the authors to show that the luckiest hyper-parameter vector is not just an over fitting to the validation/test set. Unfortunately, none of [23, 66, 24, 25, 27, 76, 29, 31] reported such distribution

620 characteristics other than the minimum value e'_{i^*,j^*} .

Furthermore, such a use of 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 grading. This practice could hardly be acceptable to any test agencies and any agencies that will use the test scores for admission purpose since this submitted error e'_{i^*,j^*} misleads due to its lack of validation.
625

The error-backprop training tends to locally fit each network on the training set T ; while the Post-Selection picks the luckiest network with parameter vector \mathbf{a}_{i^*} and initial weights \mathbf{w}_{j^*} that has the best luck on T' . If an unobserved data set T'' , disjoint with T' , $T' \cap T'' = \emptyset$, is observed from the same distribution S , the error rate e''_{i^*,j^*} of $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ is predicted to be significantly higher than e'_{i^*,j^*} ,

$$e''_{i^*,j^*} \gg e'_{i^*,j^*} \quad (14)$$

because Eq. (12) depends on the test set T' in the post selection from many networks. Of course, handling a new test is also challenging for a human student but a human learning involves learning invariant rules. Any PSUTS is a technically flawed protocol.

PSUTS is tempting especially when test sets are available to the authors of paper.
630 During all error-backprop related paper reviews I have not yet found a case in which the authors did not admit that they used USUTS when I asked. The second author of [27] claimed to have used PSUVS through a personal email to me but the first author who probably performed the experiments did not claim the same. No authors of [27] responded to PubPeer questions towards [27].

635 Weng 2020 [77, 78] argued that the claims by some public speakers that such misleading errors have approached or even succeeded human performance [61] are controversial, since there are no explicit competition rules that ban test sets to be used for Post-Selections.

2.13. Human PSUTS

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

Definition 6 (Human PSUTS). *After planning experiments 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.*

645 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 and collect the test set T' . The key to an acceptable test score lies in how much detail the human designer can plan for what is in the test sets and how much freedom s programmer has in hand picking features.

650 Poggio et al. [79] and Fukushima et al. [17] explicitly admitted their use of human PSUTS. Li Fei-Fei et al. [19] only vaguely admitted their use of human PSUTS by a vague term “weakly supervised” using an extension of formulation by Pietro Perona that is originally unsupervised. Questions raised towards [19] on PubPeer were not answered by the authors.

655 3. Why Error-Backprop Needs PSUTS

This section discusses a global view, which is new as far as the author is aware, about why error-backprop suffers from local minima even in the easier batch-learning mode.

660 Since error-backprop does not perform well for incremental learning mode as we can see why also from the following discussion, we will concentrate on batch learning mode. Namely, we let the network “see” the entire training set T for each network update.

Let us first consider a well-known neuronal model that is applicable to many CNNs. Suppose a post-synaptic neuron with activation z_j is connected to its pre-synaptic neurons y_i , $i = 1, 2, \dots, n$, through synaptic weights w_{ij} , by the expression:

$$\phi\left(\sum_{i=1}^n w_{ij} y_i\right) = z_j \quad (15)$$

where $\phi(y) = \frac{1}{1+e^{-y}}$ is the logistic function. The gradient of z_j with respect to weight vector $\mathbf{w}_j = (w_{1,j}, w_{2,j}, \dots, w_{n,j})$ is

$$\eta(y_1, y_2, \dots, y_n) \triangleq \eta \mathbf{y}$$

where η is the partial derivative of $\phi(y)$. Thus, according to gradient direction, the change of the weight vector \mathbf{w}_j is along the direction of pre-synaptic input vector \mathbf{y} . If the error is negative, z_j should increase. Then the weight vector should be incremented by

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + w_2 \mathbf{y} \quad (16)$$

where w_2 is the learning rate. We use the w_2 to relate better the optimal Hebbian learning, called LCA, used by DN in Section 4. At this point, the following theorem is in order.

665

Theorem 4 (Lacks of error-backprop). *Error-backprop lacks (1) energy conservation, (2) an age-dependent learning rate, and (3) competition based role-determination.*

670

Proof: Proof of (1): If pre-synaptic input vectors $\{\mathbf{y}\}$ are similar, multiple applications of Eq. (16) add many terms of $\{w_2 \mathbf{y}\}$ into the weight vector \mathbf{w}_j causing it to explode, which means a lack of energy conversation. Proof of (2): w_2 is typically tuned by an *ad hoc* way, such as a handpicked small value turned by a term called momentum, instead of being automatically determined in Maximum Likelihood optimality (ML-optimality) by neuronal firing age to be discussed in Section 4. Proof of (3): Suppose neuron z_j is in a hidden area of the network hierarchy. This neuron z_j updates its pre-synaptic weight using Eq. (16) regardless z_j is role-responsible or not for the current network error. Likewise, looking upstream, there is also a lack of role-determination in the gradient-based update for pre-synaptic neurons y_1, y_2, \dots, y_n , all of which must update their own weights using their own gradients. Namely, there is no competition-based role-determination in error-backprop . ■

675

The meaning (3) of Theorem 4 are illustrated by Fig. 2. CNNs do not have a competition mechanism in any 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 characterized by its

680

685

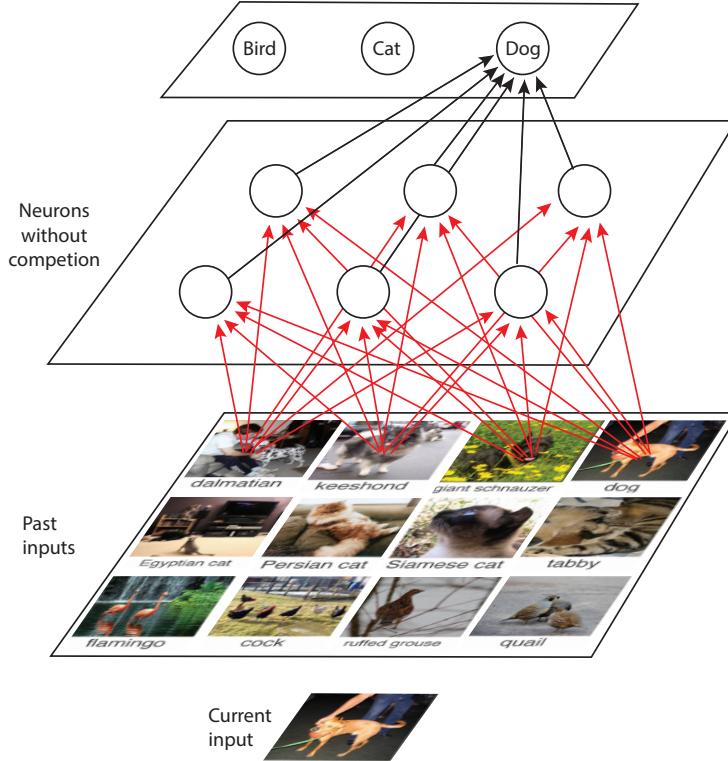


Figure 2: Lack of role-determination in hidden neurons due to a lack of competition. The same ideas are true for a deeper hierarchy. Color sample images courtesy of [61].

label “neurons without competition”. The hidden neurons in this area do not have a competition mechanism which would, like in LCA [67], allow a small proportion of neurons to win the competition and fire so that they automatically take the roles that they happened to compete well. This analysis leads us to the following theorem.

Theorem 5 (Random roles in error-backprop). *A set of random initial weights in a network assigns random roles to all hidden neurons, from which a local minimal point based on error-backprop learning inherits this particular random-role assignment. Which neurons in each hidden area take a role does not matter; but how hidden neurons share a set of roles in each hidden area does matter in the final fitting error, validation error, and test error after error-backprop.*

Proof: Without loss of generality, suppose a maximum in the output neuron means a positive classification and weights take positive and negative values. Then, a positive weight to an output neuron z_j from a hidden neuron y_i means an excitatory role of y_i to z_j and a negative weight means an inhibitory role. A zero weight means an irrelevant role. The gradient vector computed in Eq. (16) means such excitatory-inhibitory input patterns from pre-synaptic neurons are added through iterative error-backprop procedures. Because of the complete connections and an identical neuronal type, where a hidden neuron is located in the Fig. 2 does not matter, but each input image must have a sufficient number of hidden neurons in every hidden area to excite for its signals to reach the corresponding output neuron. The initial role assignment patterns in initial weights do matter for the final fitting error rate, the validation error rate, and the test error rate, because gradient updates are local and inherited such initial roles. ■

Theorem 6 (Percentage luck of error-backprop). Suppose a CNN has $l > 1$ areas, A_0, A_1, \dots, A_l , connected by a cascade or a variation thereof. A_0 takes input frames $\{\mathbf{x} \in X\}$ and A_l is the output area for classification. Suppose an area has a total of m hidden neurons that share a common receptive field R in A_0 . Consider a given input frame \mathbf{x} . Let the percentage of the m hidden neurons that do not fire among all neurons in the same area with the same receptive field be denoted as $p(\mathbf{x})$. Then, the error-backprop depends on the average $\bar{p} = E_{\mathbf{x} \in X} \{p(\mathbf{x})\}$ to be a reasonably small value, called the percentage luck.

Proof: To guide the proof, we should mention that DNs use top-k competition so that each receptive field in each area has only k neurons that fires, where k is small, e.g., top-1, for each receptive field R . Suppose a receptive field R represents a neuronal column that has n neurons. A neuronal area at level l is denoted as A_l . Every receptive field image $\mathbf{x} \in X = A_0$ is concrete by which we means that its neurons are only pixels $\{x\}$ of a concrete example of a class C with $\bar{p}(x) = P(x \text{ fires}) \approx 50\%$ (e.g., 50% black and 50% white). Each neuron z in area A_l is abstract by which we means that it fires means an abstract class C that \mathbf{x} belongs to, with $\bar{p}(z)$ being small corresponds to top-1 among n neurons. Then, it is necessary for the CNN to convert the most concrete representation of pixels in A_0 to more abstract representations in $A_l, l > 0$ with a low

$\bar{p}(z)$. For example, in Fig. 2, we have $l = 2$ and there is no completion in the hidden area A_1 . Then error-backprop depends on that each neuron in A_2 has only a relatively smaller percentage among $n = 6$ neurons in A_1 that are positive, i.e., as the features of its particular class. The requirement of being a small percentage is due to the need for other non-firing neurons to deal with many other patterns in the same receptive field.

■

As we can expect, such a low percentage condition is rarely satisfied by a random weight vector. The more random weight vectors one uses, the better chance to hit the luck.

From Theorems 4 through 6 and their proofs, we can see that the luck of role assignment is a critical flaw of error-backprop , and so are the system parameters and the simple-minded regularization of the learning rate. Because of these key reasons, PSUTS plays a critical role to select the luckiest network from many unlucky ones after error-backprop . The more networks have trained by error-backprop , the more likely the luckiest one has a good role-assignment to start with.

There has been no lack of papers that claim to justify error-backprop does not over fit, e.g., variance based stochastic gradient decent [80], saddle-free deep network [81], drop out [82], implicit regularization during gradient flow [83]. They all address only local issues of neural networks trained by error-backprop and did not mention Post-Selections. The theory here addresses, the global role-assignment problem of random weights that no local mechanisms can deal with. This seems to be why PSUTS is necessary by error-backprop , but PSUTS is controversially fraudulent in terms of protocol—test sets are meant to test a reported system, not supposed to be used to decide which network to report from many.

4. How a DN Avoids Post-Selections

Apparently, a brain does not use Post-Selection at al, whether UPSVS or PUUTS, because every human child must develop in a human environment to make his living. He should not be covered up and not reported, regardless how well or bad he performs. Cresceptron in the 1990s [42, 43, 16, 44, 45] and later DN [46, 84, 57, 58] were inspired

by the interactive mode that brains learn though lifetime. In other words, Cresceptron and DN do not need Post-Selections. Furthermore, every DN must be ML-optimal given the same Three Learning Conditions.

⁷⁶⁰ 4.1. New AI Metrics: Developmental Errors

In contrast to Post-Selections likely used by [21, 14, 19, 79, 23, 66, 24, 25, 27, 76, 29, 31] including probably AlphaGo [26], AlphaGo Zero [28], AlphaZero [85], AlphaFold [30] and MuZero [86] and many others, we define and reported developmental errors that includes all errors occurred through lifetime of each learning network:

Definition 7 (Developmental error). A Developmental Network is denoted as $N = (X, Y, W_y, Z, W_z, A)$ with sensory area X , skull closed hidden area Y and its weight space W_y , and motor area Z and its weight space W_z , and the space of architecture parameters A , where X , Y , and Z also denote the spaces of responses of X , Y and Z areas, respectively. The space of architecture parameters A includes all remaining parameters and memory of the network, other than neuronal weights, such as ages of neurons (for learning rates), neuronal patterning parameters (location and receptive fields adapted by synaptic maintenance), neuronal types (for initial connection absences among areas), and neuronal growth rates (for speed of mitosis). It runs through lifetime by sampling at discrete time indices as $N(t)$, $t = 0, 1, 2, \dots$. Start at inception $t = 0$ with supervised sensory input $\mathbf{x}_0 \in X(0)$, initial state $\mathbf{z}_0 \in Z(0)$, randomly initialized weigh vector $\mathbf{y}_0 \in Y(0)$, initial architecture $\mathbf{a}_0 \in A(0)$. At each time t , $t = 1, 2, \dots$, the network $N(t)$ recursively and incrementally updates:

$$(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{a}_t) = f(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1}, \mathbf{a}_{t-1}) \quad (17)$$

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 $N(t)$ generates a motor vector \mathbf{z}_t and is observed by the teacher and its vector difference from the desired \mathbf{z}_t^* is recorded as error e_t . The lifetime average error for each motor concept or component, from time 0 up to time t is defined

as

$$\bar{e}(t) \triangleq \frac{1}{t} \sum_{i=0}^t e_i, \quad (18)$$

which is computed incrementally in terms of average developmental error $\bar{e}(t)$:

$$\bar{e}(t) = \frac{t-1}{t} \bar{e}(t-1) + \frac{1}{t} e_t. \quad (19)$$

765 Namely, all errors across a lifetime, at every time instance, are caught by the developmental error. In order to reach a small error, a low final error rate that a batch learning method tries to reach is not sufficient. Instead, the network must learn as fast as possible and avoid errors as much as possible at every time instance t . This is indeed important since earlier performance will shape later learning.

770 An optimal network that gives the lowest possible developmental error, among all possible networks under the same Three Learning Conditions, must be optimal at every time instance t throughout its life. DN is one such network. Post-Selections are useless among neural networks that give the smallest developmental error under the same Three Learning Conditions, because the maximum-likelihood optimality should give 775 equivalent networks of the same developmental error.

However, in practice, the learning experience in the Three Learning Conditions is unlikely the same among different networks, because each physical robot that runs a network at least occupies distinct physical locations in the real world. For example, if two physical robot in the same family fight for a toy, the winner gains a winner 780 experience and the loser may acquire a loser mentality. In other words, even if the parents of two boys are not biased toward any boys, the competition among the boys results in different learning experiences.

The developmental error is important. If a competition is based on developmental errors (such as during AIML Contests [62]), the winner is unlikely be one that uses a 785 brute force method but has an excessive amount of computational resources and manpower. ImageNet competitions [61] are flowed also in this sense.

Although not formally defined as developmental errors, Cresceptron [16] and Developmental Networks [41, 87, 88] reported developmental errors.

Namely, the developmental error, unless stated otherwise for a particular time period, 790 is the average lifetime error from inception. To report more detailed information

about the process of developmental errors $\{\mathbf{e}_t | t \geq 0\}$, statistics other than the mean (average) can be utilized, such as the minimum, 25%, 50% (median), 75%, the maximum, and the standard deviation.

For more a specific time period, such as the period from age t_1 to age t_2 , the average
795 error is denoted as $\bar{e}[t_1 : t_2]$. Therefore, $\bar{e}(t)$ is a short notation for $\bar{e}[0 : t]$.

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

$$\mathbf{c}_t \triangleq (\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \quad (20)$$

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 free-state neuron that happens to be the best match is spawned that perfectly
memorizes this new context regardless its randomly initialized weights. When all the
800 free 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 [41] and
for DN-2 by [87], as we will discuss below.

Note that a developmental system has two input areas from the environment, sensory X and motor Z . That is, motor Z is supervisable by the world (including teachers)
805 but not often. 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 make an error, the motor-error that the teacher
made is also recorded as the developmental error of the motor of the learner but due to
810 the teacher.

4.2. Neuronal Competitions

As discussed above, error-backprop learning is without neuronal competitions. The
main purpose of competition is to automatically assign roles to hidden neurons. Below,
we consider two kinds of Convolution Neural Networks (CNNs), sensory networks and
815 sensorimotor networks. A sensory network is feedforward, from sensor to motor, in
computation flow and therefore is simpler and easier to understand. A sensorimotor

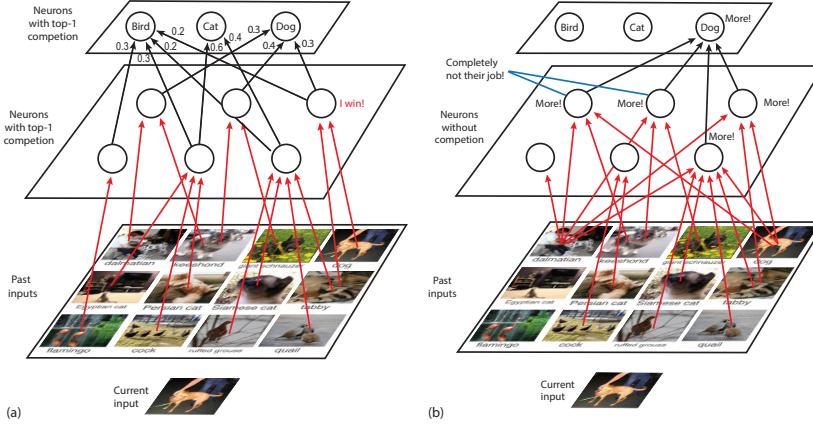


Figure 3: How competition automatically assigns roles among hidden neurons without a central controller: The case for automatically construct a mapping $f : X \mapsto L$. (a) The number of samples in X is larger than the number of hidden neurons such that each hidden neuron must win-and-fire for multiple 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 [61].

network takes both sensor and motor as inputs and is highly recurrent and therefore more powerful.

4.2.1. Sensory networks

Let us first consider the case of feed-forward networks as illustrated in Fig. 3. Fig. 3(a) shows a situation where the number of samples in X is larger than the number of hidden neurons, which is typical. 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 $(\mathbf{x}, l) \in X \times L$ arrives incrementally through time, $t = 0, 1, 2, \dots$. This is the case with Cresceptron (and some other networks) which conducts incremental learning by dealing with image-label pairs one at a time and update the network 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 (also called drop-out).

835 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. 3(a) enables each hidden neuron to respond to multiple features, indicated by the
 840 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. 3(a), $k = 1$
 for top- k competition in Y .

Likewise, suppose top-1 competition in the next higher layer, Y , namely each time
 845 only one Y neuron fires at 1 and all other Y neurons do not fire, resulting the connection
 patterns from the second layer Y to the next higher layer Z . In the output layer Z , top-1
 competition takes place but a human teacher can supervise the pattern.

The Candid Covariance-free Incremental (CCI) Lobe Component Analysis (LCA)
 850 in Weng 2009 [67] 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 com-
 ponent 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 LCA vector.

Intuitively, regardless what random weights each hidden neuron starts with, as soon
 as it wins to fire for the first time, 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
 $w_2 = 1/a = 1$ so that the new weight vector becomes the first input $r\mathbf{x}$ with $r = 1$ for
 the firing winner.

$$\mathbf{v} \leftarrow (1 - \frac{1}{a})\mathbf{v} + \frac{1}{a}r\mathbf{x}. \quad (21)$$

855 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 4.

Fig. 3(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.).
 865 The CCI LAC guarantees that the sum of weights for each Z neuron sum to 1. This automatic role assignment optimally solves the random role problem of error-backprop in Theorem 5.

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

4.2.2. Sensorimotor networks

The main reason that Marvin Minsky [5] 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
 875 (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. 3, but also a sensorimotor feature. By sensorimotor feature, we mean that the firing of each hidden neuron in Fig. 3 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
 880 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.

4.3. FA as sensorimotor mapping

This sensorimotor feature is easier to understand if we use the conventional symbols for (symbolic) automata. 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

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

AFA as a control of any Turing machine. Weng 2015 [41] extended the definition the
885 FA so that it outputs its state so the resulting FA becomes an Agent FA (AFA). Further,
Weng 2015 [41] extended the action q to the machinery of Turing machine (see Fig. 4)
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 [41] proved that
the control of any Turing machine is an AFA, a surprising result.

890 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.

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

$$\begin{bmatrix} Z(0) \\ Y(0) \\ X(0) \end{bmatrix} \rightarrow \begin{bmatrix} Z(1) \\ Y(1) \\ X(1) \end{bmatrix} \rightarrow \begin{bmatrix} Z(2) \\ Y(2) \\ X(2) \end{bmatrix} \rightarrow \dots \quad (22)$$

where \rightarrow means neurons on the right use the input neurons on the right and compete
895 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. With LCA update, [41] proved that such a DN is ML-optimal and has a constant complexity for each update $O(1)$ with a large constant, suited for real-time computation with a large memory and many neurons.

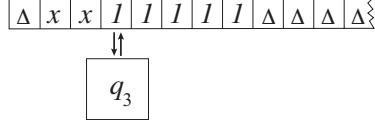


Figure 4: A Turing machine has a tape, a read-write head, and a transition function with a current state.

900 4.4. DN as a ML-Optimal of Emergent Universal Super-Turing Machine

The traditional Turing Machine (TM) is a human handcrafted machine, as illustrated in Fig. 4.

A Universal TM (UTM) is 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 905 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 of the current general-purpose computers because the user can write any program on any set of appropriate data for the UTM to carry out. Because 910 a DN is an ML-optimal emergent FA, Weng 2015 [41] 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. The programs on 915 the tape are learned by the Super UTM incrementally from the real world across its lifetime!

4.5. DN as a ML-Optimal Learning Engine for APFGP and Conscious Learning

Because DN is an ML-optimal learning engine for any TM, including UTM, DN 920 ML-optimally learns any UTM from the physical world, conditioned on those in Definition 2. This means that a DN ML-optimally learns to Autonomous Programming for General Purposes (APFGP) [57, 89]. Based on the capability of APFGP, Weng 2020 argued that APFGP is a characterization of conscious machines [58] that boots its skill of consciousness through conscious learning—being (partially) conscious while learning across lifetime. Hopefully, APFGP is a clearer and more precise characterization

925 for conscious machines and animals, assuming that we allow a conscious machine to develop its degree of consciousness from infancy.

In the following, we list the DN algorithm so that we can understand APFGP is not a vague idea and how APFGP by DN avoids Post-Selections.

4.6. DN-2 Algorithm

930 Let us go through the DN-2 algorithm here so that we can see that DN is fully detail in computer implementation.

DN-2 is the latest general-purpose learning engine in the DN family. In DN-1, the allocation of neurons in each subarea of the hidden Y area is handcrafted by the designer. In DN-2, several biology-inspired mechanisms are added to automatically 935 allocate neuronal resources and generate a dynamic and fluid hierarchy of internal representations during learning, relieving the human designer from handcrafting a concept hierarchy, beyond the rigid hierarchy in deep learning [42, 43, 16, 44, 45, 19, 79, 23, 66, 24, 25, 27, 76, 29, 31]. Namely, a DN-2 starts with simple internal representations which gradually grow to be rich and deep supported by early representations as a “brain 940 stem”, but it is still ML-optimal conditioned on those in Definition 2.

Areas from low to high: X : sensory; Y hidden (internal); Z : motor. From low to high: bottom-up. From high to low: top-down. From one area to the same area: lateral. X does not link with Z directly.

945 Input areas: X and Z ; Output areas: Z ; Hidden area: Y , fully closed from $t = 0$.

1. At time $t = 0$, inception. Initialize the X , Y and Z areas. $\mathbf{x} \in X$ takes the first image. Set every Y neuron with random weights, zero firing age, and zero response $\mathbf{y}(0)$. Set the total number of Y neurons to be n_Y . A boundary c_Y indicates the number of active neurons ($c_Y \leq n_Y$). Set the Z area and its memory part M_Z similarly, but all concept zones take none vectors if the learner has no 950 prenatally learned inborn “reflexes”.
2. For time $t = 1, 2, \dots$, repeat the following steps forever (executing steps 2a, 2b in parallel, before step 2c):

(a) All Y neurons compute in parallel:

$$(\mathbf{y}', M'_Y) = f_Y(\mathbf{c}_Y, M_Y) \quad (23)$$

where context $\mathbf{c}_Y = (\mathbf{x}, \mathbf{y}, \mathbf{z})$, M_A denotes the memory of area A including weights and neuronal firing ages, and f_Y is the Y area function using LCA [65, 67]. If the best active Y neurons do not match the input vector well, area Y transfers new neurons to active and increment the boundary c_Y .

(b) Supervise \mathbf{z}' if the teacher likes. Otherwise, Z neurons compute the response vector \mathbf{z} and update memory M'_Z in parallel:

$$(\mathbf{z}', M'_Z) = f_Z(\mathbf{c}_Y, M_Z) \quad (24)$$

where f_Z is the Z area function using LCA [65, 67] and $\mathbf{c}_Z = (\mathbf{y}, \mathbf{z})$.

(c) Replace asynchronously: $(\mathbf{y}, M_Y, \mathbf{z}, M_Z) \leftarrow (\mathbf{y}', M'_Y, \mathbf{z}', M'_Z)$. Supervise input \mathbf{x} .

The area function f_Y in Eq.(23) and area function f_Z in Eq.(24) include two parts:

- (1) The computation of response vectors \mathbf{y}' and \mathbf{z}' , respectively; (2) The maintenance of memory M'_Y and M'_Z for Y area and Z area, respectively.

The ML-optimality of DN-1 and DN-2 is rooted in the optimality of LCA and extends to the entire network and entire lifetime.

4.7. Methods for Recursive Optimization

Given the Three Learning Conditions, at each time t , $t = 1, 2, \dots$, 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, there $\theta(t) = (\mathbf{w}_y, \mathbf{w}_z, \mathbf{a})$ where some parameters of the architecture parameter vector \mathbf{a} are hand-initialized such as the receptive fields. The maximum estimator for θ corresponds to the θ that maximizes the probability density. Regardless

\mathbf{z} is imposed, \mathbf{z} is part of the parameters to be computed in a closed-form as a self-generated version:

$$(\theta^*, \mathbf{y}^*, \mathbf{z}^*) = \underset{(\theta, \mathbf{y}, \mathbf{z})}{\operatorname{argmax}} f_\theta(\mathbf{x}, \mathbf{z}). \quad (25)$$

Since the above 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 in a closed-form by the incremental version of Eq. (25) where f uses context $\mathbf{c}_{t-1} = (\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1})$:

$$(\theta_t^*, \mathbf{y}_t^*, \mathbf{z}_t^*) = \underset{(\theta_t, \mathbf{y}_t, \mathbf{z}_t)}{\operatorname{argmax}} f_{\theta_t}(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}^*, \mathbf{z}_{t-1}^*). \quad (26)$$

- 970 The DN computes the above expression for each time t in a closed form without conducting any iterations [41, 87].

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 $V = \{\dot{\mathbf{v}}_i | i = 1, 2, \dots, k + 1\}$ is the current synaptic vectors in Y . Whenever the network takes an input \mathbf{p} , compute 975 the pre-responses in Y . If the top-1 winner in Y has a pre-response lower than almost perfect match $m(t)$ discussed below, activate a free neuron to fire. Eq. (21) showed that the initial weights of this free neuron is multiplied by a zero and therefore do not affect its updated weights.

Weng [41] proved that DN-1 computes the ML-estimator of all observations from 980 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. Weng [87] did the same for DN-2.

4.8. How DN Avoids Post-Selections but is Further ML-Optimal

Since weights are initialized randomly, how does a DN result in an equivalent network regardless the random seed? There are $k + 1$ initial neurons in the Y area, and $V = \{\dot{\mathbf{v}}_i | i = 1, 2, \dots, k + 1\}$ is the current synaptic vectors in Y . Whenever the network takes an input \mathbf{p} , every Y neuron computes the pre-response. If the top-1 winner in Y has a pre-response lower than almost perfect match $m(t)$, activate a free neuron

to fire. The almost perfect match $m(t)$ is defined as follows:

$$m(t) = (1 - \delta)(1 - e^{-t/t_1}) \quad (27)$$

985 where δ is the bound of machine round-off errors, and t_1 the childhood length.

Using a mathematical induction procedure, Weng [41] proved that DN-1 computes the ML-estimator of all observations from $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ using a large constant time complexity for each time t . Weng et al. 2018 [87] proved the ML-optimality for DN-2. Since the number of transition of any Turing machine is finite, when the DN learns a 990 Turing machine, a finite number of hidden Y neurons is sufficient for the DN to incrementally memorize exactly all the transitions observed from the Turing machine. In other words, although DN learns incrementally, such a DN is error-free for learning any complex Turing machines, including any universal Turing machines.

If the DN runs in the real world, any finite size DN is not error-free soon after 995 inception since the number of observations from the real world is virtually unbounded, although each life is time bounded. Namely, the amount of data from the real world is so large that any practically large DN will eventually run out of free neurons that have not fired yet. From that point on, the DN is no longer guaranteed to be error-free, although could be sometimes error-free, but is still ML-optimal inside the skull 1000 conditioned on those in Definition 2. In other words, in the sense of ML, the DN is free of local minima inside the skull. That is why only one DN is sufficient for each life and the DN avoids Post-Selections. However, because the three conditions in Definition 2, a DN is not be optimal unconditionally either. For example, a better designed teaching schedule or a more appropriate physical environment may enable a DN to learn and 1005 discover rules faster and better.

4.9. Comparison with HMM

It is important to compare the traditional Hidden Markov Model (HMM) [90] with the ML-optimal DN. (1) The former does not have any internal representations other than the symbol based probabilities; the latter self-generates internal representations to 1010 generalize based on internal-representation based probabilities (e.g., weights). (2) The former uses batch learning but the latter uses incremental learning. (3) States in the

former are symbolic, static, only partially observable for HMM, and not teachable but those in the latter are emergent, observable and directly teachable if the teacher like. (4) The former requires a batch clustering method (e.g., k-mean clustering) to initialize a
1015 static set of symbolic states, but the states/actions in the latter are incrementally taught or autonomously generated and tried. (5) Clusters of states in the former are not supported by a statistical optimality and the probability is only for state estimates but those in the latter are ML-optimal throughout the lifetime of learning, not in states/actions that the learner must produce and do not have a freedom for, but for the internal representations that the learner does have a high degree of freedom for. (6) Due to the need
1020 to compute internal representations, the amount of computations in the latter is often higher than the former but the computational complexity is linear in time with a large constant (the number of weights of all available neurons).

5. Experiments

1025 5.1. Vision, Audition and Natural Languages

The recent experimental results of DN work here include (1) vision that includes simultaneous recognition and detection and vision-guided navigation on MSU campus walkways [91], (2) audition to learn phonemes with a simulated cochlea and the corresponding behaviors [92], (3) acquisition of English and French in an interactive
1030 bilingual environment [93], and (4) exploration in a simulated maze environment with autonomous learning for vision, path cost, planning, and selection of the least-cost plan, where all such emergent actions are either *covert* (thoughts) or *overt* (acts) [94]. The same network was used to learn these four very different tasks and task environments while each task embeds the ML-optimality of the network, under the Three Learning
1035 Conditions.

5.2. Error-Backprop vs. ML-Optimal DN

To show the effects of the absence of ML-optimality in CNN vs. the ML-optimality of DN, Fig. 5 shows the errors of the luckiest Convolutional Neural Network (CNN) trained by a batch error-backprop method and the errors of a DN trained incrementally.

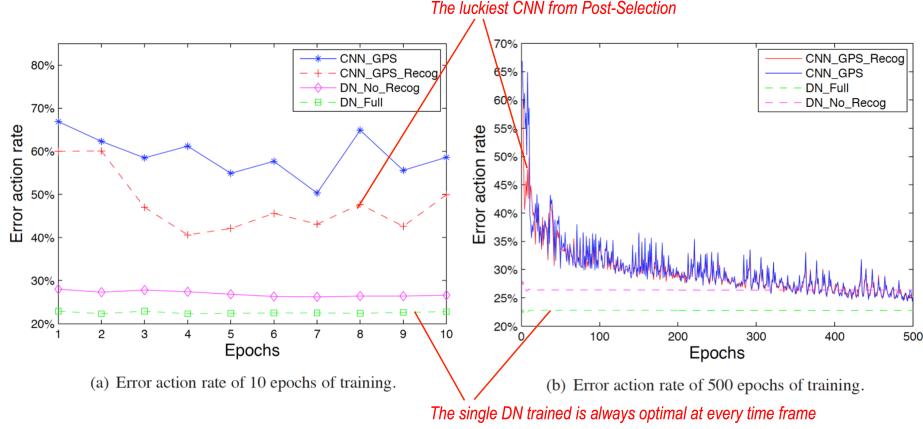


Figure 5: Comparison of error between the luckiest CNN trained by batch error-backprop and a DN across different epochs through the training data. “Recog” and “Full” means teach where-what rules. Otherwise, data-fitting only. Adapted from [95].

1040 As we understand, batch learning should not be compared with an incremental learning method, because it is not a comparison on an equal footing. However, Fig. 5 shows that
 1045 DN does a harder (incremental) work drastically better than CNN does an easier (batch) work. The task is real-world vision-guided navigation on the campus of Michigan State University. Because the DN is optimal in maximum-likelihood, it reaches the minimum
 1050 error as soon as it has gone through the data set T once (one epoch). Later epochs correspond to reviews of the same data set T . According to the maximum-likelihood principle, the optimal estimate of the neuronal weights should not change but the ages of the neurons continue to advance. In contrast, the luckiest error-backprop trained CNN chosen from several random seeds need many epochs to reduce its errors and
 1055 only very slowly. At the end of 500th epoch, the error of the luckiest CNN trained by error-backprop is still considerably higher than the full DN. Furthermore, as shown in Fig. 5, teaching invariant concepts, i.e., abstraction in Theorem 2, are used for reducing the optimal errors. For more detail, the reader is referred to [95].

5.3. AIML Contests

1055 In the AIML Contest 2016, all teams are required to use a single learning engine to learn three sensory modalities, vision, audition, and bilingual natural languages acqui-

sition, while the engine learns in “lifetime”. Although all the teams are free to choose any existing learning engines such as DN, tensor-flow or other engines, all the teams chose DN engine (open source). The supplied simulated sequential data (yes, subject to the “big data” flaw) are as follows. When we let the x be the image at each time instance and z be the pattern of landmark location-and-type and action of navigation, the DN became a vision-guided navigation machine. When we let x be the frame of firing pattern of hair cells in cochlea at each time instance and z be the dense states getContexts and the sparse type of sounds, the DN became a auditory-recognizer machine. When we let x be a time frame of vector of word (either English or French) and z be the language kind (neutral, English, and French) and meaning of each sentence context, the DN became a bilingual language learner and recognizer. Thus, the AIML Contest appeared to be the first contest that independently demonstrated task-nonspecificity and modality-nonspecificity by independent laboratories with the contest teams. All the teams are evaluated under the same Three Learning Conditions, e.g., the number of neurons in the engine must not exceed the same given bound. The Contest used the developmental error like the one defined here averaged over all the three contest tasks and across the three lifetimes. The *developmental error* ranked all the submitted contest entries and required all networks not to exceed the specified maximum number of neurons for each task so that the competition does not unfairly favor those teams that have more computational resources at their disposal but not necessarily that their methods are more superior. All the teams have a high degree of freedom to modify the learning engine and to modify the supplied motor actions on the given data set, such as generating attentive actions on the given training set to train invariant concepts (e.g., where and what concepts) which modifies the default training experience supplied by the AIML Contest organizers but was still based on the same supplied data set. The prudent design of the AIML contests was meant to avoid the corresponding problems in ImageNet Contests [61] and many other contests.

5.4. GENISAMA Applications

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. Hopefully, as a APFGP platform, this new kind of human-wearable robots will be useful for practitioners to produce various kinds of intelligent auto-programed software. The author predicts that such a new kind of AI
1090 systems will considerably alleviate the high brittleness of traditional AI software and traditional robot software in open and natural world.

Hopefully, future DN-driven robots will learn consciously and autonomously discover in the real world for Turing machine based general purposes, with relative infrequent interactions from humans similar to what parents do to their children and
1095 human teachers teach their students in classrooms. The experiments and competitions described here are for this grand goals but have not reached this experimental goal yet.

6. Conclusions

We used intuitive terms but formal ways to discuss Post-Selections. Public and media have gained an impression that deep learning has approached or even “sometimes
1100 exceeded” human level performance on certain tasks. For example, the image classification errors from a static image set were compared with those of humans [61, A2, p242]) and the work is laudable. However, this paper raises Post-Selections, which seem to question such claims since a real human does not have the luxury of Post-Selections. The author hopes that the exposure of Post-Selections is beneficial to AI
1105 credibility and the future healthy development of AI, especially with the concepts of developmental errors and the framework of ML-optimal lifetime learning for invariant concepts under the Three Learning Conditions. Some researchers have raised that it seems that those who wan a competition were those who have more computational resources and manpower at their disposal. The new developmental error metrics under the Three Learning Conditions hopefully encourages future AI competitions to compare methods under the same Three Learning Conditions. Considering DN as a much-simplified model for a biological machine, it seems not baseless to guess that each biological brain is probably ML-optimal (of course in a much richer sense) across lifetime, e.g., due to the pressure to compete at every age. The Three Learning Conditions explicitly include other factors that greatly affect machine learning performances
1115

such as learning framework (e.g., task-nonspecificity, incremental learning, the robot bodies), learning experiences and computational resources. The analysis that any “big data” sets are nonscalable does not mean that we should not create, use and share data sets. Instead, we need to pay attention to the fundamental limitations of any static data sets, regardless how large their apparent sizes are.

References

- [1] N. Montfort, *Twisty Little Passages: An Approach to Interactive Fiction*, MIT Press, Cambridge, MA, 2005.
- [2] A. M. Turing, Computing machinery and intelligence, *Mind* 59 (1950) 433–460.
- [3] J. Weng, Symbolic models and emergent models: A review, *IEEE Trans. Autonomous Mental Development* 4 (1) (2012) 29–53.
- [4] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Edition, Prentice-Hall, Upper Saddle River, New Jersey, 2010.
- [5] M. Minsky, Logical versus analogical or symbolic versus connectionist or neat versus scruffy, *AI Magazine* 12 (2) (1991) 34–51.
- [6] D. B. Lenat, G. Miller, T. T. Yokoi, CYC, WordNet, and EDR: Critiques and responses, *Communications of the ACM* 38 (11) (1995) 45–48.
- [7] L. Gomes, Machine-learning maestro Michael Jordan on the delusions of big data and other huge engineering efforts, *IEEE Spectrum*.
- [8] D. E. Rumelhart, J. L. McClelland, the PDP Research Group, *Parallel Distributed Processing*, Vol. 1, MIT Press, Cambridge, Massachusetts, 1986.
- [9] J. L. McClelland, D. E. Rumelhart, The PDP Research Group (Eds.), *Parallel Distributed Processing*, Vol. 2, MIT Press, Cambridge, Massachusetts, 1986.
- [10] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. J. Li, D. A. Shamma, M. S. Ernstein, L. Fei-Fei, Visual genome, *International Journal of Computer Vision* 123 (1) (2017) 32–73.

- [11] K. I. Funahashi, On the approximate realization of continuous mappings by neural networks, *Neural Network* 2 (2) (1989) 183–192.
- [12] T. Poggio, F. Girosi, Networks for approximation and learning, *Proceedings of the IEEE* 78 (9) (1990) 1481–1497.
- [13] T. Kohonen, *Self-Organizing Maps*, 3rd Edition, Springer-Verlag, Berlin, 2001.
- [14] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biological Cybernetics* 36 (1980) 193–202.
- [15] M. Oja, S. Kaski, T. Kohonen, Bibliography self-organizing maps (som) papers: 1998 - 2001 addendum, *Neural Computing Surveys* 3 (2003) 1–156.
- [16] J. Weng, N. Ahuja, T. S. Huang, Learning recognition and segmentation using the Cresceptron, *International Journal of Computer Vision* 25 (2) (1997) 109–143.
- [17] K. Fukushima, S. Miyake, T. Ito, Neocognitron: A neural network model for a mechanism of visual pattern recognition, *IEEE Trans. Systems, Man and Cybernetics* 13 (5) (1983) 826–834.
- [18] T. Serre, T. Poggio, M. Riesenhuber, L. Wolf, S. Bileschi, High-performance vision system exploiting key features of visual cortex, US Patent US7606777B2.
- [19] L. Fei-Fei, R. Fergus, P. Perona, One-shot learning of object categories, *IEEE Trans. Pattern Analysis and Machine Intelligence* 28 (4) (2006) 594–611.
- [20] J. Weng, Dialog initiation: Modeling and: Closed skull or not?, *IEEE CIS Autonomous Mental Development Newsletter* 9 (2) (2012) 10–11.
- [21] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, Wiley, Chichester, 1994.
- [22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of IEEE* 86 (11) (1998) 2278–2324.

- [23] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems 25, 2012, pp. 1106–1114.
- 1170 [24] Y. LeCun, L. Bengio, G. Hinton, Deep learning, *Nature* 521 (2015) 436–444.
- [25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, *Nature* 518 (2015) 529–533.
- 1175 [26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search, *Nature* 529 (2016) 484–489.
- 1180 [27] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwinska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. C., H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, D. Hassabis, Hybrid computing using a neural network with dynamic external memory, *Nature* 538 (2016) 471–476.
- 1185 [28] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, *Nature* (2017) 354–359.
- 1190 [29] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafiyan, T. Back, M. Chesus, G. S. Corrado, A. Darzi, M. Etemadi, F. Garcia-Vicente, F. J. Gilbert, M. Halling-Brown, D. Hassabis, S. Jansen, A. Karthikesalingam, C. J. Kelly, D. King, J. R. Ledsam, D. Melnick, H. Mostofi, L. Peng, J. J. Reicher, B. Romera-Paredes, R. Sidebottom, M. Suleyman, D. Tse,

- 1195 K. C. Y. and Jeffrey De Fauw, S. Shetty, International evaluation of an AI system
for breast cancer screening, *Nature* 577 (2020) 89–94.
- [30] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin,
A. Zidek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Si-
monyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, D. Hass-
abis, Improved protein structure prediction using potentials from deep learning,
Nature 577 (2020) 706–710.
- [31] M. G. Bellemare, S. Candido, J. G. P. S. Castro, M. C. Machado, S. Moitra,
S. S. Ponda, Z. Wang, Autonomous navigation of stratospheric balloons using
reinforcement learning, *Nature* 588 (7836) (2020) 77–82.
- 1200 [32] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, J. Clune, First return, then
explore, *Nature* 590 (7847) (2021) 580–586.
- [33] V. Saggio, B. E. Asenbeck, A. Hamann, T. Stromberg, P. Schiansky, V. Dun-
jko, N. Friis, N. C. Harris, M. Hochberg, D. Englund, S. Wolk, H. J. Briegel,
P. Walther, Experimental quantum speed-up in reinforcement learning agents, *Na-
ture* 591 (7849) (2021) 229–233.
- 1210 [34] F. R. Willett, D. T. Avansino, L. R. Hochberg, J. M. Henderson, K. V.
Shenoy, High-performance brain-to-text communication via handwriting, *Nature*
593 (7858) (2021) 249–254.
- [35] N. Slonim, Y. Bilu, C. Alzate, R. Bar-Haim, B. Bogin, F. Bonin, L. Choshen,
E. Cohen-Karlik, L. Dankin, L. Edelstein, L. Ein-Dor, R. Friedman-Melamed,
A. Gavron, A. Gera, M. Gleize, S. Gretz, D. Gutfreund, A. Halfon, D. Hersh-
covich, R. Hoory, Y. Hou, S. Hummel, M. Jacovi, C. Jochim, Y. Kantor, Y. Katz,
D. Konopnicki, Z. Kons, L. Kotlerman, D. Krieger, D. Lahav, T. Lavee, R. Levy,
N. Liberman, Y. Mass, A. Menczel, S. Mirkin, G. Moshkowich, S. Ofek-Koifman,
M. Orbach, E. Rabinovich, R. Rinott, S. Shechtman, D. Sheinwald, E. Shnarch,
I. Shnayderman, A. Soffer, A. Spector, B. Sznajder, A. Toledo, O. Toledo-Ronen,
E. Venezian1, R. Aharonov, An autonomous debating system, *Nature* 591 (7850)
(2021) 379–384.

- [36] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le1, J. Laudon, R. Ho, R. Carpenter, J. Dean, A graph placement methodology for fast chip design, *Nature* 594 (7862) (2021) 207–212.
- [37] M. Y. Lu, T. Y. Chen, D. F. K. Williamson, M. Zhao, M. Shady, J. Lipkova, F. Mahmood, AI-based pathology predicts origins for cancers of unknown primary, *Nature* 594 (7861) (2021) 106–110.
- [38] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Handler, P. Pickkers, N. A. Aziz, S. Ktena, F. Tran, M. Bitzer, S. Ossowski, N. Casadei, C. Herr, D. Petersheim, U. Behrends, F. Kern, T. Fehlmann, P. Schommers, C. Lehmann, M. Augustin, J. Rybniker, J. Altmuller, N. Mishra, J. P. Bernardes, B. Kramer, L. Bonaguro, J. Schulte-Schrepping, E. D. Domenico, C. Siever, M. Kraut, M. Desai, B. Monnet, M. Saridaki, C. M. Siegel, A. Drews, M. Nuesch-Germano, H. Theis, J. Heyckendorf, S. Schreiber, S. Kim-Hellmuth, C. A. S. (COVAS), J. Nattermann, D. Skowasch, I. Kurth, A. Keller, R. Bals, P. Nurnberg, O. Rieb, P. Rosenstiel, M. G. Netea, F. Theis, S. Mukherjee, M. Backes, A. C. Aschenbrenner, T. Ulas, D. COVID-19 Omics Initiative (DeCOI), M. M. B. Breteler, E. J. Giamparellos-Bourboulis, M. Kox, M. Becker, S. Cheran, M. S. Woodacre, E. L. Goh, J. L. Schultze, Swarm learning for decentralized and confidential clinical machine learning, *Nature* 594 (7862) (2021) 265–270.
- [39] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, E. Theilen, Autonomous mental development by robots and animals, *Science* 291 (5504) (2001) 599–600.
- [40] J. L. McClelland, K. Plunkett, J. Weng, Guest editorial: Convergent approaches to the understanding of autonomous mental development, *IEEE Trans. on Evolutionary Computation* 11 (2) (2007) 133–136.
- [41] J. Weng, Brain as an emergent finite automaton: A theory and three theorems, *International Journal of Intelligent Science* 5 (2) (2015) 112–131.

- [42] J. Weng, N. Ahuja, T. S. Huang, Cresceptron: a self-organizing neural network which grows adaptively, in: Proc. Int'l Joint Conference on Neural Networks, Vol. 1, Baltimore, Maryland, 1992, pp. 576–581.
- [43] J. Weng, N. Ahuja, T. S. Huang, Learning recognition and segmentation of 3-D objects from 2-D images, in: Proc. IEEE 4th Int'l Conf. Computer Vision, 1993, pp. 121–128.
- [44] J. Weng, Life is science (35): Did Turing Awards go to plagiarism?, Facebook blog, www.facebook.com/juyang.weng/posts/10158305658699783 (March 4 2020).
- [45] J. Weng, Did Turing Awards go to plagiarism?, YouTube Video, 1:05 hours, <https://youtu.be/EAhkH79TKFU> (May 27 2020).
- [46] J. Weng, Why have we passed “neural networks do not abstract well”?, Natural Intelligence: the INNS Magazine 1 (1) (2011) 13–22.
- [47] Z. Ji, J. Weng, D. Prokhorov, Where-what network 1: “Where” and “What” assist each other through top-down connections, in: Proc. IEEE Int'l Conference on Development and Learning, Monterey, CA, 2008, pp. 61–66.
- [48] Q. Guo, X. Wu, J. Weng, Cross-domain and within-domain synaptic maintenance for autonomous development of visual areas, in: Proc. the Fifth Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics, Providence, RI, 2015, pp. +1–6.
- [49] D. J. Felleman, D. C. Van Essen, Distributed hierarchical processing in the primate cerebral cortex, Cerebral Cortex 1 (1991) 1–47.
- [50] C. M. Super, Environmental effects on motor development: A case of Africa infant precocity, Developmental Medicine and Child Neurology 18 (1976) 561–567.
- [51] K. A. Thoroughman, J. A. Taylor, Rapid reshaping of human motor generalization, Journal of Neuroscience 25 (39) (2005) 8948–8953.

- 1280 [52] G. Rizzotti, L. Riggio, I. Dascola, C. Umita, Reorienting attention across the horizontal and vertical meridians: evidence in favor of a premotor theory of attention, *Neuropsychologia* 25 (1987) 31–40.
- [53] T. Moore, K. M. Armstrong, M. Fallah, Visuomotor origins of covert spatial attention, *Neuron* 40 (2003) 671–683.
- 1285 [54] J. M. Iverson, Developing language in a developing body: the relationship between motor development and language development, *Journal of child language* 37 (2) (2010) 229–261.
- [55] J. Weng, M. Luciw, Brain-like emergent spatial processing, *IEEE Trans. Autonomous Mental Development* 4 (2) (2012) 161–185.
- 1290 [56] J. Weng, M. Luciw, Q. Zhang, Brain-like temporal processing: Emergent open states, *IEEE Trans. Autonomous Mental Development* 5 (2) (2013) 89 – 116.
- [57] J. Weng, Autonomous programming for general purposes: Theory, *International Journal of Humanoid Robotics* 17 (4) (2020) 1–36.
- 1295 [58] J. Weng, Conscious intelligence requires developmental autonomous programming for general purposes, in: Proc. IEEE International Conference on Development and Learning and Epigenetic Robotics, Valparaiso, Chile, 2020, pp. 1–7.
- [59] J. Weng, On post selections using test sets (PSUTS) in AI, in: Proc. International Joint Conference on Neural Networks, Shenzhen, China, 2021, pp. 1–8.
- 1300 [60] J. Weng, A developmental method that computes optimal networks without post selections, in: Proc. IEEE International Conference on Development and Learning, Beijing, China, 2021, pp. 1–6.
- [61] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, *International Journal of Computer Vision* 115 (2015) 211–252.

- [62] J. Weng, Z. Zheng, X. Wu, J. Castro-Garcia, S. Zhu, Q. Guo, X. Wu, Emergent Turing machines and operating systems for brain-like auto-programming for general purposes, in: Proc. AAAI 2018 Fall Symposium: Gathering for Artificial Intelligence and Natural Systems, Arlington, Virginia, 2018, pp. 1–7.
- 1310 [63] D. H. Ballard, C. M. Brown, Computer Vision, Prentice-Hall, New Jersey, 1982.
- [64] L. Shapiro, G. Stockman, Computer Vision, Addison-Wesley, New York, 2001.
- [65] J. Weng, Natural and Artificial Intelligence: Introduction to Computational Brain-Mind, 2nd Edition, BMI Press, Okemos, Michigan, 2019.
- 1315 [66] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in: Proc. Computer Vision and Pattern Recognition, Columbus, Ohio, 2014, pp. +1–8.
- [67] J. Weng, M. Luciw, Dually optimal neuronal layers: Lobe component analysis, IEEE Trans. Autonomous Mental Development 1 (1) (2009) 68–85.
- 1320 [68] J. Weng, M. D. Luciw, Brain-inspired concept networks: Learning concepts from cluttered scenes, IEEE Intelligent Systems Magazine 29 (6) (2014) 14–22.
- [69] D. J. Wood, J. S. Bruner, G. Ross, The role of tutoring in problem-solving, Journal of Child Psychology and Psychiatry (1976) 89–100.
- [70] S. Burr, Active learning literature survey, Data Mining and Knowledge Discovery 2 (2) (1998) 121–167.
- 1325 [71] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Communications of the ACM 60 (6) (2017) 84–90.
- [72] A. K. Jain, R. C. Dubes, Algorithms for Clustering Data, Prentice-Hall, New Jersey, 1988.
- 1330 [73] Y. Wang, X. Wu, J. Weng, Synapse maintenance in the where-what network, in: Proc. Int'l Joint Conference on Neural Networks, San Jose, CA, 2011, pp. 2823–2829.

- [74] Q. Guo, X. Wu, J. Weng, WWN-9: Cross-domain synaptic maintenance and its application to object groups recognition, in: Proc. Int'l Joint Conference on Neural Networks, Beijing, China, 2014, pp. +1–8.
- 1335 [75] Q. Gao, G. A. Ascoli, L. Zhao, BEAN: Interpretable and efficient learning with biologically-enhanced artificial neuronal assembly regularization, *Front. Neuro-robot* (2021) 1–13.
- 1340 [76] M. Moravcik, M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, Deepstack: Expert-level artificial intelligence in heads-up no-limit poker, *Science* 356 (2017) 508–513.
- [77] J. Weng, Life is science (36): Did Turing Awards go to fraud?, Facebook blog, www.facebook.com/juyang.weng/posts/10158319020739783 (March 8 2020).
- [78] J. Weng, Did Turing Awards go to fraud?, YouTube Video, 1:04 hours, <https://youtu.be/Rz6CFIKrx2k> (June 4 2020).
- 1345 [79] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, T. Poggio, Robust object recognition with cortex-like mechanisms, *IEEE Trans. Pattern Analysis and Machine Intelligence* 29 (3) (2007) 411–426.
- [80] P. Sermanet, K. Kavukcuoglu, S. Chintala, Y. LeCun, No more pesky learning rates, in: Proc. International Conference on Machine learning, Atlanta, GA, 2013, pp. 343–351.
- 1350 [81] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, in: Advances in Neural Information Processing Systems, Curran Associates, Inc., Montreal, Canada, 2014, pp. 2933–2941.
- 1355 [82] N. Srivastava, G. E. Hinton, K. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research* 15 (1) (2014) 1929–1958.

- [83] T. Poggio, Theoretical issues in deep networks, *Proceedings of the National Academy of Sciences* 117 (48) (2020) 30039–30045.
- 1360 [84] J. Weng, Z. Zheng, X. Wu, J. Castro-Garcia, Auto-programming for general purposes: Theory and experiments, in: Proc. International Joint Conference on Neural Networks, Glasgow, UK, 2020, pp. 1–8.
- [85] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, *Science* (2018) 1140–1144.
- 1365 [86] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, D. Silver, Mastering atari, go, chess and shogi by planning with a learned model, *Science* 588 (7839) (2020) 604–609.
- 1370 [87] J. Weng, Z. Zheng, X. Wu, Developmental Network Two, its optimality, and emergent Turing machines, U.S. Provisional Patent Application Serial Number: 62/624,898, published (Feb. 1 2018).
- [88] J. A. Knoll, V. N. Hoang, J. Honer, S. Church, T. H. Tran, J. Weng, Optimal developmental learning for multisensory and multi-teaching modalities, in: Proc. IEEE International Conference on Development and Learning, Beijing, China, 2021, pp. 1–6.
- 1375 [89] J. Weng, A unified hierarchy for AI and natural intelligence through auto-programming for general purposes, *Journal of Cognitive Science* 21 (2020) 53–102.
- 1380 [90] L. R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, *Proceedings of IEEE* 77 (2) (1989) 257–286.
- [91] Z. Zheng, X. Wu, J. Weng, Emergent neural turing machine and its visual navigation, *Neural Networks* 110 (2019) 116–130.

- 1385 [92] X. Wu, J. Weng, Muscle vectors as temporally “Dense Labels”, in: Proc. International Joint Conference on Neural Networks, Glasgow, UK, 2020, pp. 1–8.
- [93] J. Castro-Garcia, J. Weng, Emergent multilingual language acquisition using developmental networks, in: Proc. International Joint Conf. Neural Networks, IEEE Press, Budapest, Hungary, 2019, pp. +1–8.
- 1390 [94] X. Wu, J. Weng, On machine thinking, in: Proc. International Joint Conf. Neural Networks, IEEE Press, Shenzhen, China, 2021, pp. 1–8.
- [95] Z. Zheng, J. Weng, Mobile device based outdoor navigation with on-line learning neural network: a comparison with convolutional neural network, in: Proc. 7th Workshop on Computer Vision in Vehicle Technology (CVVT 2016) at CVPR 2016, Las Vega, 2016, pp. 11–18.

Declaration of interests

- The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.
- The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

None

On Post Selection Using Test Sets (PSUTS) in AI

Juyang Weng

Department of Computer Science and Engineering, Cognitive Science Program, Neuroscience Program
Michigan State University, East Lansing, MI, 48824 USA
GENISAMA LLC, 4460 Alderwood Drive, Okemos, Michigan 48864 USA

Abstract—This is a theory paper. It first raises a rarely reported but unethical practice in Artificial Intelligence (AI) called Post Selection Using Test Sets (PSUTS). Consequently, the popular error-backprop methodology in deep learning lacks an acceptable generalization power. All AI methods fall into two broad schools, connectionist and symbolic. PSUTS practices have two kinds, machine PSUTS and human PSUTS. The connectionist school received criticisms for its “scruffiness” due to a huge number of scruffy parameters and now the machine PSUTS; but the seemingly “clean” symbolic school seems more brittle than what is known because of using human PSUTS. This paper formally defines what PSUTS is, analyzes why error-backprop methods with random initial weights suffer from severe local minima, why PSUTS violates well-established research ethics, and how every paper that used PSUTS should have at least transparently reported PSUTS data. For improved transparency in future publications, this paper proposes a new standard for AI metrology, called developmental errors for all networks trained in a project that the selection of the luckiest network depends on, along with Three Conditions: (1) system restrictions, (2) training experience and (3) computational resources.

I. INTRODUCTION

In 1950 Alan Turing published his now celebrated paper [1] titled *Computing Machinery and Intelligence*. Turing [1] was impressive to have discussed a wide variety of considerations for machine intelligence, as many as nine categories. Unfortunately, he suggested to consider what is now called the Turing Test that has inspired and misled many AI researchers.

Much progress has been made in AI since 1950 and many methods have been developed to deal with AI problems. For the scope of this paper, we will focus on generalization. All AI methods fall into two schools, symbolic and connectionist, although many detailed methods are a mixture of both.

A. Symbolic school

Symbols are used in many AI methods (e.g., states in HMMs, nodes in Graphical Models and attributes in SLAM), since they are intuitive to human programmers. However, symbols are static and have some fundamental limitations that have not received sufficient attention.

The symbolic school [2] assumes a micro-world in 4D space-time in which a set of objects or concepts, e.g., $S = \{o_1, o_2, \dots, o_n\}$, is assumed to be uniquely defined among human programmers and their computers, represented by a series of symbols in time $\{o_1(t), o_2(t), \dots, o_n(t) \mid t_0 \leq t < t_1\}$. The correspondences among all these symbols $\{o_i\}$ of the same object across different times are known as “the frame problem” [2] in AI which means that the programmer must

manually link all such symbols together along time for each object. In computer vision, the symbolic school assumes a single symbol o_i , for all its 3D positions in its 3D trajectory $\{\mathbf{x}(t) \mid t_0 \leq t \leq t_1\}$ and uses techniques, such as feature tracking through video (e.g., for driverless cars). Therefore, the symbolic school is based on human-handcrafted set of symbols and their assumed meanings. Marvin Minsky wrote that symbols are “neat” [3], but in fact, symbols are “neat” mainly in human programmer’s understanding but not in relating computer programs to a real world. A major problem that many AI researchers did not address is the generalization weakness of the symbolic school, as the following definition states:

Definition 1 (Symbolic Brittleness): Suppose a symbolic AI machine $M(S)$ designed for a handcrafted set S of symbols is applied to a real world that requires a new set of symbols S' , with $S' \neq S$. $M(S')$ fails without a human programmer that handcrafts a proper mapping between every element in S' to an element in S .

Many expert systems (e.g., CYC, WordNet and EDR) and many big-data projects require a human programmer to be in the loop during deployment due to this symbolic brittleness. Because it is extremely challenging for a human programmer to understand many implicit limitations of $M(S)$, the mapping that the human establishes typically makes $M(S')$ fail, resulting in the well-known high brittleness of symbolic systems. This author also did some early work that belongs to the symbolic school [4] whose weakness motivated him to start in 1990 a new departure with Cresceptron discussed below which seems to be the first deep learning network because Fukushima’s deep network Neocognitron with performance evaluation [5] (that this author cited) did not learn (i.e., humans handpicked features to be discussed below as human PSUTS). It is worth noting that earlier verisons of Neocognitron 1975 1980, both in *Biological Cybernetics* did not have performance evaluation and therefore did not report the need for human PUSTS during performance evaluation.

The developmental methods to be discussed below are supposed to automatically address this problem without a need for a human programmer to be in the loop of handcrafting a mapping. Below, this author will argue that the symbolic school and some methods in the connectionist school suffer from human PSUTS.

B. Connectionist School

The connectionist school claimed to be less brittle. However, a network is egocentric—meaning that the agent starts from its own (neural) network, instead of a symbolic world set S . It must learn from the external *world* without a handcrafted, *world-centered* object model S . Although connectionist methods often assume some task-specific symbols, e.g., a static set S of object labels, they also assume a restricted world implicitly. Therefore, a connectionist model typically needs to sense and learn from a restricted world using a network. The use of S in any neural networks as a set of object labels is a fundamental limitation that also causes the resulting system to be brittle for the same reason as the symbolic school.

Typically, a network is meant to establish only a mapping f from the space of input X to the space of object class S ,

$$f : X \mapsto S \quad (1)$$

Many video analysis problems, speech recognition problems, and computer game-play problems are also converted into this static input space X by converting temporal attributes into components of the static input space X .

Two main types of learning algorithms have been typically used, human handpicking of features [5]–[8] (i.e., skull-open) and error backprop [6], [9]–[11] (i.e., skull-closed). Below, this author will argue that all neural networks that use error backprop suffer from machine PSUTS.

By the way, *genetic algorithms* offer another approach to such network learning. These algorithms study changes in genomes across different lives, but many genetic algorithms do not deal with lifetime development [12]. This author suggested that handcrafting functions of a genome as a Developmental Program (DP) seems to be a clean and tractable problem which avoids the extremely high cost of evolution on DP. Many genetic algorithms further suffer from the PSUTS problems, since they often use test sets as training sets (i.e., vanished tests) as explained below.

Marvin Minsky [3] complained among many scholars that neural networks are “scruffy”. This problem is not addressed completely until the framework of Emergent Turing Machine was introduced [13] into Developmental Networks (DNs) by the Developmental School discussed below. A lack of Emergent Turing Machine logic or being “scruffy” is the main cause of PSUTS in neural networks trained by error backprop methods.

C. Developmental School

The main thrust of the Developmental School, formally presented 2001 by Weng and six co-authors [12] is the task-nonspecificity for lifetime development, known as Developmental Programs (DPs). Although a DP generates a neural network, a DP is very different from a conventional neural network in evaluation of performance across each life—all errors from the inception time 0 of each life is recorded and reported up to each current time $t > 0$, as explained further below.

The first developmental program seems to be the Cresceptron by Weng et al. [14]. As Neocognitron does not learn, Cresceptron appears to be also the first deep-learning Convolutional Neural Network (CNN). Cresceptron seems to be the first incremental neural network whose evaluation of performance is across its entire “life” and only one network was generated and tested for reporting error rates cross the entire life. Cresceptron did not deal with time. A developmental approach that deals with space and time in a unfired fashion using a neural network started from Developmental Networks (DNs) [15] whose experimental embodiments range from WWN-1 to WWN-9. The DNs went beyond vision problems to attack general AI problems including vision, audition, and natural language acquisition as emergent Turing machines [13]. DNs overcame the limitations of the framewise mapping in Eq. (1) by dealing with lifetime mapping:

$$f : X(t-1) \times Z(t-1) \mapsto Z(t), t = 1, 2, \dots \quad (2)$$

where $X(t)$ and $Z(t)$ are the sensory input space and motor input-output space, respectively, and \times denotes the Cartesian product of sets. Note that $Z(t-1)$ here is extremely important since it corresponds to the state of a Turing machine. Namely, all the errors occurred during any time of each life is recorded and taken into account in the performance evaluation. Different from the space mapping in Eq. (1) and very important, the space $Z(t)$ is the directly teachable space for the learning system, inspired by brains [16], [17], [17]–[20]. This new formulation is meant to model not only brain’s spatial processing [21] and temporal processing [22], but also Autonomous Programming for General Purposes (APFGP) [23], [24]. Based on the APFGP capability, the AI field seems to have a powerful yet general-purpose framework towards conscious machines [25].

In the following, we will discuss what PSUTS is in Section II. Section III addresses why error backprop algorithms suffer from severe local minima problems. Section IV proposes a new kind of error, called developmental error that addresses the problems with PSUTS. Section V provides concluding remarks.

II. PSUTS

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

Definition 2 (The Three Conditions): The Three Conditions for developing an AI system are: (1) the system 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. Task-specific vs. Task-nonspecific

Task-nonspecific learning and task-specific learning differ greatly as explained in Weng et al. 2001 [12]. In a task-specific paradigm, the system developer is given a task e.g., constructing a driverless car. Then, it is the human programmer who chooses a world model, such as a model of lane edges. Next, he picks an algorithm based on this world model, e.g., using the well-known Hough transform algorithm that makes every pixel detected as edge cast votes for lines of all possible orientations that go through the pixel. Then the top-two “peaks” of line parameters that have received the highest votes are used to declare two lanes detected from the image. Some lane-tracking assistance systems use this lane-model-based approach in the symbolic school. Here “edge” and “two lanes” are two symbolic concepts picked up by the programmer. Such systems will fail when lanes are unclear or totally disappear due to weather or road conditions, leading to a brittle system. Human brains appear to be more resilient.

In contrast, a task-nonspecific approach [12] not only avoids any symbolic model, but also does not assume that a task is given. The desirable actions at any time are recalled automatically by the learner based on system’s learned context [26] that represents automatically figured-out task and appropriate context of the task—like a brain. Thanks to the absence of any world model, such as lanes, this task-nonspecific approach has a potential to be more robust than a world-model-based approach. A task-nonspecific approach typically uses a neural network to learn [12].

B. Batch vs. Incremental learning modes

Neural network learning has two learning modes, batch learning and incremental learning.

With batch learning, a human first collects a set D of data (e.g., images) and then labels each datum with a desirable output (e.g., command of navigation or class label). A neural network is trained to approximate a mapping $f : X \mapsto Q$ where X is the space of all possible images and Q is the set of outputs. Many batch-learning projects use the error-backprop method [10], [11], [27].

As we will discuss in Section III, the error backprop mechanism erases important long-term memories along the gradient direction if the learning mode is incremental. Thus, few networks that use error backprop on a large data set adopt a incremental learning mode.

In contrast, all developmental methods cited here use incremental learning mode for long lifetimes, since new neurons are incrementally activated into the network. The competition guarantees that the winner is the most appropriate neuron whose memory is the current working memory [28].

The batch and incremental learning modes are not capability-equivalent [28]. The former requires all sensory inputs are available at a batch, independent of the corresponding actions. This is incorrect according to sensorimotor recurrence. By sensorimotor recurrence, we mean that sensory inputs and motor outputs are mutually dependent on each other in a recurrent way. We have the following theorem:

Theorem 1 (Big-Data Flaw): All big-data sets used by a machine learning method violate the *sensorimotor recurrence* property of the real world.

Proof: A learning agent at time $t - 1$, as shown in Eq. (2) does not have the next sensory input from $X(t)$ available before the corresponding actions in $Z(t-1)$ are generated and outputted, since the sensory input in $X(t)$ varies according to the agent actions in $Z(t - 1)$. As an example, turning head left or right will result in a different image sensed. Therefore, all static big-data sets violate the sensorimotor recurrence. ■

All methods that use PSUTS below violate the sensorimotor recurrence, because they use a static set of training data. Therefore, it is inappropriate for any of them to claim near-human performance since human learning is incremental due to the sensorimotor recurrence during a human’s lifetime.

There are two kinds of PSUTS, machine PSUTS and human PSUTS.

C. Machine PSUTS

The available data set D is divided into three mutually disjoint sets, a training set T , a validation set V , and a test set T' . Two sets are disjoint if they do not share any elements. Let us consider how machine PSUTS arises from experiments.

A network architecture has a set of parameters represented by a vector, where each component corresponds to a 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 parameter vectors need to be tried, $A = \{\mathbf{a}_i \mid i = 1, 2, \dots, k\}$. If there are 10 parameters and each of which has 10 grid points to try, there are a total of $k = 10^{10} = 10B$ architecture parameter vectors to try, an extremely large number.

As we will see in Section III, given any architecture parameter vector \mathbf{a}_i , it is unlikely that a single network initialized by a set of random weight vectors can give an acceptable error rate on the training set, called fitting error, that error backprop intends to minimize. That is how the multiple sets of random weight vectors come in. For each architecture vector \mathbf{a}_i , assume n sets of random weights \mathbf{w}_j , resulting in kn networks

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

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 . Grave et al. 2016 seems to have mentioned $n = 20$. Using the above example of $k = 10B$, $kn = 200B$ a huge number that requires a lot of computational resources and manpower.

We are ready to define Post Selection Using Validation Sets (PSUVS):

Definition 3 (Machine PSUVS): 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:

$$e_{i^*, j^*} = \min_{1 \leq i \leq k, 1 \leq j \leq n} e_{i,j} \quad (3)$$

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

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):

Definition 4 (Machine PSUTS): 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:

$$e'_{i^*,j^*} = \min_{1 \leq i \leq k, 1 \leq j \leq n} e'_{i,j} \quad (4)$$

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

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.

Definition 5 (Distribution of errors of trained systems):

The distributions of all kn trained networks' fitting errors $\{f_{ij}\}$, validation errors $\{e_{ij}\}$, and test errors $\{e'_{ij}\}$, $i = 1, 2, \dots, k$, $j = 1, 2, \dots, n$, 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. For transparency, it seems necessary to report such distribution characteristics other than the minimum value e'_{i^*,j^*} .

They should report such distribution characteristics to reflect the effects of machine PSUTS. 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 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^*} ,

$$e''_{i^*,j^*} \gg e'_{i^*,j^*} \quad (5)$$

because Eq. (4) depends on the test set T' in the post selection from many networks. Of course, handling new tests is also challenging for a human student but any PSUTS is unethical.

D. Human PSUTS

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

Definition 6 (Human PSUTS): 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.

Some neural network publications, e.g., [5], [8], [29] appear to have used human PSUTS.

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 and collect the test set T' . The key to an acceptable test score lies in how much detail the human designer can plan for what is in the test sets.

III. WHY ERROR BACKPROP NEEDS PSUTS

As shown by Fig. 7 of author's group's publication [30] in a real-world vision-guided navigation task using error-backprop vs. DN, even the "luckiest" batch-learning network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ using Machine PSUTS performed poorly, e.g., 60% error rate from error-backprop vs. 22% error rate from DN, both on disjoint test set T' after the first epoch going through the entire training set T once. After having gone through T 500 times, the error rates are 25% from error-backprop vs. 22% from DN, since DN has already reached the ML-optimal solution as early as the first epoch.

This section discusses about a global view, which is new as far as the author is aware, about why error backprop even for the easier batch-learning mode suffers from local minima.

Since error-backprop does not perform acceptably well for incremental learning mode, as we can see why also from the following discussion, we will concentrate on batch learning mode only. Namely, we let the network see the entire training set T for each network update.

Let us first consider a well-known neuronal model that is applicable to many CNNs. Suppose a post-synaptic neuron with activation z_j is connected to its pre-synaptic neurons y_i , $i = 1, 2, \dots, n$, through synaptic weights w_{ij} , by the expression:

$$\phi\left(\sum_{i=1}^n w_{ij} y_i\right) = z_j \quad (6)$$

where $\phi(y) = \frac{1}{1+e^{-y}}$ is the logistic function. The gradient of z_j with respect to weight vector $\mathbf{w}_j = (w_{1,j}, w_{2,j}, \dots, w_{n,j})$ is

$$\eta(y_1, y_2, \dots, y_n) \triangleq \eta \mathbf{y}$$

where η is the partial derivative of $\phi(y)$. Thus, according to gradient direction, the change of the weight vector \mathbf{w}_j is along the direction of pre-synaptic input vector \mathbf{y} . If the error is negative, z_j should increase. Then the weight vector should be incremented by

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + w_2 \mathbf{y} \quad (7)$$

where w_2 is the learning rate. We use the w_2 to relate better the optimal Hebbian learning, called LCA, used by DN in Section IV. At this point, the following theorem is in order.

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

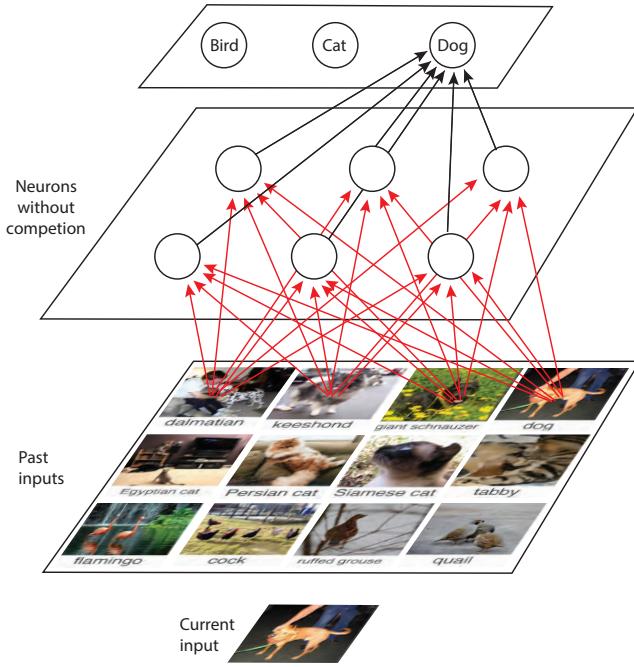


Fig. 1. Lack of role-determination in hidden neurons.

Proof: Proof of (1): If pre-synaptic input vectors $\{\mathbf{y}\}$ are similar, multiple applications of Eq. (7) add many terms of $\{w_2\mathbf{y}\}$ into the weight vector \mathbf{w}_j causing it to explode. Proof of (2): w_2 is typically tuned by an *ad hoc* way, such as a handpicked small value turned by a term called momentum, instead of being automatically determined ML-optimality by neuronal firing age to be discussed in Section IV, Eq.(11). Proof of (3): Suppose neuron z_j is in a hidden area of the network hierarchy. This neuron z_j updates its pre-synaptic weight using Eq. (7) regardless z_j is role-responsible or not for the network error. Likewise, there is a lack of role-determination in the area of y_1, y_2, \dots, y_n , all of which must update using their own gradients. Namely, there is a competition-based role-determination in error backprop. ■

The meaning (3) of Theorem 2 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. This analysis leads us to the following theorem.

Theorem 3 (Random Roles in Error-BackProp): A set of random initial weights in a network assigns random roles to

all hidden neurons, from which a local minimal point based on error-backprop learning inherits this particular random-role assignment. Which neurons in each hidden areas take a role does not matter, but how hidden neurons share a set of roles in each hidden area does matter.

Proof: Without loss of generality, suppose a maximum in the output neuron means a positive classification and weights take positive and negative values. Then, a high weight to an output neuron z_j from a hidden neuron y_i means an excitatory role to z_j and a low weight means an inhibitory role. A zero weight means an irrelevant role. The gradient vector computed in Eq. (7) means such excitatory-inhibitory input patterns from pre-synaptic neurons are added through iterative error-backprop procedures. Because of the complete connections and identical neurons, where a hidden neuron is located in the Fig. 1 does not matter, but each input image must have a sufficient number of hidden neurons in every hidden area to reach the corresponding output neuron. The role assignment patterns in initial weights do matter in terms of the fitting error rate, the validation error rate, and the test error rate. ■

Theorem 4 (Percentage Luck of Error-BackProp):

Suppose a CNN has $l > 1$ areas, A_0, A_1, \dots, A_l , connected by a cascade or a variation thereof. A_0 takes input frames $\{\mathbf{x} \in X\}$ and A_l is the output area for classification. It has a total of m hidden neurons in an area that share a common receptive field R in A_0 . Let the percentage of the m hidden neurons do not fire given an input frame \mathbf{x} be denoted as $p(\mathbf{x})$. Then, the Error-BackProp depends on the average $\bar{p} = E_{\mathbf{x} \in X} \{p(\mathbf{x})\}$ to be a reasonably small value, called the percentage luck.

Proof: To guide the proof, we should mention that DNs use top-k competition so that each receptive field in each area has only k neurons that fires, where k is small, e.g., top-1, for each receptive field R . Every receptive field image $\mathbf{x} \in X$ is concrete by which we means that its neurons are only pixels of a concrete example of a class C with $\bar{p} \approx 50\%$. Each neuron in A_l is abstract by which we means that it fires means an abstract class C that \mathbf{x} belongs to, with \bar{p} corresponds to top-1. Then, it is necessary for the CNN to convert the most concrete representation in A_0 with a high \bar{p} to the most abstract A_l with a low \bar{p} . For example, in Fig. 1, we have $l = 2$ and there is no completion in the hidden area A_1 . Then error-backprop depends on that each neuron in A_2 has only few weights from the 6 neurons in A_1 that are positive, i.e., as its features. ■

From Theorems 2 through 4 and their proofs, we can see that depending on the luck of role assignment is a critical flaw of error-backprop, and so are the system parameters and the simple-minded regularization of the learning rate. Because of these key reasons, PSUTS plays a critical role to select the luckiest network from many unlucky ones after error backprop. The more networks have trained by error backprop, the more likely the luckiest one has a good role-assignment to start with.

There has been no lack of papers that claim to justify error backprop does not over fit, e.g., variance based stochastic gradient descent [31], saddle-free deep network [32], drop out [33], implicit regularization during gradient flow [34].

They all address only local issues of neural networks trained by error backprop and did not mention PSUTS. The theory here addresses, for the first time in a journal submission, the global role-assignment problem of random weights that no local arguments can deal with. That seems to be why PSUTS is necessary by error backprop, but PSUTS is controversially fraudulent in terms of research ethics—test sets are meant to test a reported system, not are supposed to be used to decide which network to report from many.

IV. DEVELOPMENTAL ERRORS

Apparently, brains do not use PUSTS, as every human child must normally develop in a human environment to make the living. Cresceptron [14] and later DN [15] were inspired by the interactive mode that brains learn though lifetime.

A. Developmental Errors

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

Definition 7 (Developmental error): 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 $t = 0, 1, 2, \dots$ 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 $t = 1, 2, \dots$, the network N recursively and incrementally updates:

$$(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{m}_t) = f(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1}, \mathbf{m}_{t-1}) \quad (8)$$

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

$$\bar{e}(t) \triangleq \frac{1}{t} \sum_{i=0}^t e_i. \quad (9)$$

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

$$\mathbf{c}_t \triangleq (\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \quad (10)$$

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 in [13]. 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, $t = 0, 1, 2, \dots$. 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).

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

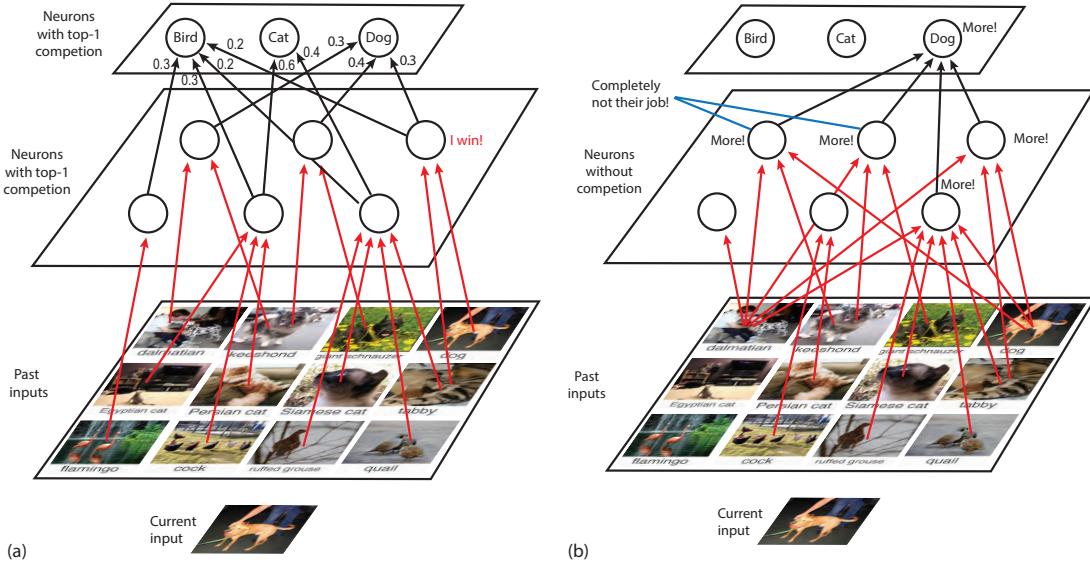


Fig. 2. How competition automatically assigns roles among hidden neurons without a central controller: The case for automatically construct a mapping $f : X \mapsto S$. (a) Each hidden neuron must win-and-fire for multiple inputs. (b) Error-backprop from the “dog” motor neuron asks some hidden neurons to help but the current input feature is not their job.

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 [28] 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.

$$\mathbf{v} \leftarrow (1 - \frac{1}{a})\mathbf{v} + \frac{1}{a}r\mathbf{x}. \quad (11)$$

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 2.

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.

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 [3] 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 [13].

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. FA as sensorimotor mapping

This sensorimotor feature is 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

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

Weng [13] extended the definition an FA so that it outputs its state so the resulting FA becomes an Agent FA (AFA). Further, Weng [13] extended the action q to the machinery of Turing machine 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 [13] proved that any Turing machine is an AFA.

Weng [13] proved that a DN is ML-optimal and has a low time complexity $O(1)$ suited for real-time computation. The ML-optimality is conditioned on the Three Conditions: (1) task-nonspecific and incremental learning, (2) a training experience, and (3) a limited number of hidden neurons. Namely, under the same Three Conditions, every DN is performance equivalent, free of local minima; one network is sufficient.

V. CONCLUSIONS

Although public has received claims that deep learning with error backprop has approached or even exceeded human level performance on certain tasks (e.g., classification for static images), this paper raises PSUTS which seems to question such claims. The author hopes that the exposure of PSUTS improves AI credibility and its future development. Fig. 7 of [30] experimentally showed the ML-optimal DN leads to a much smaller developmental error by training only a single network although the ML-optimality is not to minimize it.

REFERENCES

- [1] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, pp. 433–460, October 1950.
- [2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice-Hall, 3rd ed., 2010.
- [3] M. Minsky, “Logical versus analogical or symbolic versus connectionist or neat versus scruffy,” *AI Magazine*, vol. 12, no. 2, pp. 34–51, 1991.
- [4] J. Weng, T. S. Huang, and N. Ahuja, *Motion and Structure from Image Sequences*. New York: Springer-Verlag, 1993.
- [5] K. Fukushima, S. Miyake, and T. Ito, “Neocognitron: A neural network model for a mechanism of visual pattern recognition,” *IEEE Trans. Systems, Man and Cybernetics*, vol. 13, no. 5, pp. 826–834, 1983.
- [6] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Chichester: Wiley, 1994.
- [7] T. Serre, T. Poggio, M. Riesenhuber, L. Wolf, and S. Bileschi, “High-performance vision system exploiting key features of visual cortex,” *US Patent*, vol. US7606777B2, Sept. 1 2006.
- [8] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.
- [9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [10] A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.
- [11] Y. LeCun, L. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, 2015.
- [12] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen, “Autonomous mental development by robots and animals,” *Science*, vol. 291, no. 5504, pp. 599–600, 2001.
- [13] J. Weng, “Brain as an emergent finite automaton: A theory and three theorems,” *International Journal of Intelligent Science*, vol. 5, no. 2, pp. 112–131, 2015.
- [14] J. Weng, N. Ahuja, and T. S. Huang, “Learning recognition and segmentation using the Cresceptron,” *International Journal of Computer Vision*, vol. 25, pp. 109–143, Nov. 1997.
- [15] J. Weng, “Why have we passed “neural networks do not abstract well”?” *Natural Intelligence: the INNS Magazine*, vol. 1, no. 1, pp. 13–22, 2011.
- [16] C. M. Super, “Environmental effects on motor development: A case of Africa infant precocity,” *Developmental Medicine and Child Neurology*, vol. 18, pp. 561–567, 1976.
- [17] K. A. Thoroughman and J. A. Taylor, “Rapid reshaping of human motor generalization,” *Journal of Neuroscience*, vol. 25, no. 39, pp. 8948–8953, 2005.
- [18] G. Rizzotti, L. Riggio, I. Dascola, and C. Umiltà, “Reorienting attention across the horizontal and vertical meridians: evidence in favor of a premotor theory of attention,” *Neuropsychologia*, vol. 25, pp. 31–40, 1987.
- [19] T. Moore, K. M. Armstrong, and M. Fallah, “Visuomotor origins of covert spatial attention,” *Neuron*, vol. 40, pp. 671–683, 2003.
- [20] J. M. Iverson, “Developing language in a developing body: the relationship between motor development and language development,” *Journal of child language*, vol. 37, no. 2, pp. 229–261, 2010.
- [21] J. Weng and M. Luciw, “Brain-like emergent spatial processing,” *IEEE Trans. Autonomous Mental Development*, vol. 4, no. 2, pp. 161–185, 2012.
- [22] J. Weng, M. Luciw, and Q. Zhang, “Brain-like temporal processing: Emergent open states,” *IEEE Trans. Autonomous Mental Development*, vol. 5, no. 2, pp. 89 – 116, 2013.
- [23] J. Weng, Z. Zheng, X. Wu, and J. Castro-Garcia, “Auto-programming for general purposes: Theory and experiments,” in *Proc. International Joint Conference on Neural Networks*, (Glasgow, UK), pp. 1–8, July 19-24 2020.
- [24] J. Weng, “Autonomous programming for general purposes: Theory,” *International Journal of Humanoid Robotics*, vol. 17, pp. 1–36, August 2020.
- [25] J. Weng, “Conscious intelligence requires developmental autonomous programming for general purposes,” in *Proc. IEEE International Conference on Development and Learning and Epigenetic Robotics*, (Valparaiso, Chile), pp. 1–7, Oct. 26-27 2020.
- [26] J. Weng, *Natural and Artificial Intelligence: Introduction to Computational Brain-Mind*. Okemos, Michigan: BMI Press, second ed., 2019.
- [27] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *Proc. Computer Vision and Pattern Recognition*, (Columbus, Ohio), pp. +1–8, June 24-27, 2014.
- [28] J. Weng and M. Luciw, “Dually optimal neuronal layers: Lobe component analysis,” *IEEE Trans. Autonomous Mental Development*, vol. 1, no. 1, pp. 68–85, 2009.
- [29] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, “Robust object recognition with cortex-like mechanisms,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411–426, 2007.
- [30] Z. Zheng and J. Weng, “Mobile device based outdoor navigation with on-line learning neural network: a comparison with convolutional neural network,” in *Proc. 7th Workshop on Computer Vision in Vehicle Technology (CVVT 2016) at CVPR 2016*, (Las Vega), pp. 11–18, June 269 2016.
- [31] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, “No more pesky learning rates,” in *Proc. International Conference on Machine learning*, (Atlanta, GA), pp. 343–351, June 16-21 2013.
- [32] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” in *Advances in Neural Information Processing Systems*, (Montreal, Canada), pp. 2933–2941, Curran Associates, Inc., 2014.
- [33] N. Srivastava, G. E. Hinton, K. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [34] T. Poggio, “Theoretical issues in deep networks,” *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30039–30045, 2020.

Post-Selections in AI and How to Avoid Them

Juyang Weng^{*†‡§}

^{*}Department of Computer Science and Engineering,

[†]Cognitive Science Program,

[‡] Neuroscience Program,

Michigan State University, East Lansing, MI, 48824 USA

[§] GENISAMA LLC, 4460 Alderwood Drive, Okemos, Michigan 48864 USA

Abstract: Neural network based Artificial Intelligence (AI) has reported increasing scales in experiments. However, this paper raises a rarely reported stage in such experiments called Post-Selection alter the reader to several possible protocol flaws that may result in misleading results. All AI methods fall into two broad schools, connectionist and symbolic. The Post-Selection fall into two kinds, Post-Selection Using Validation Sets (PSUVS) and Post-Selection Using Test Sets (PSUTS). Each kind has two types of post-selectors, machines and humans. The connectionist school received criticisms for its “black box” and now the Post-Selection; but the seemingly “clean” symbolic school seems more brittle because of its human PSUTS. This paper first presents a controversial view: all static “big data” are non-scalable. We then analyze why error-backprop from randomly initialized weights suffers from severe local minima, why PSUVS lacks cross-validation, why PSUTS violates well-established protocols, and why every paper involved should transparently report the Post-Selection stage. To avoid future pitfalls in AI competitions, this paper proposes a new AI metrics, called developmental errors for all networks trained, under Three Learning Conditions: (1) an incremental learning architecture (due to a “big data” flaw), (2) a training experience and (3) a limited amount of computational resources. Developmental Networks avoid Post-Selections because they automatically discover context-rules on the fly by generating emergent Turing machines (not black boxes) that are optimal in the sense of maximum-likelihood across lifetime, conditioned on the Three Learning Conditions.

Keywords: Experimental Protocols, Error-Backprop, Deep Learning, Performance Evaluation, Maximum Likelihood, Turing Machines

I. INTRODUCTION

AI research dates back at least to early 1910 when Leonardo Torres y Quevedo built a chess end game player called El Ajedrecista [1]. In 1950, Alan Turing published his now celebrated paper [2] titled *Computing Machinery and Intelligence*. Turing [2] was impressive to have discussed a wide variety of considerations for machine intelligence, as many as nine categories. Unfortunately, he suggested to consider what is now called the Turing Test that has inspired and misled many AI researchers.

Much progress has been made in AI since then and many methods have been developed to deal with AI problems. As the scope of this paper, we will focus on generalization. All AI methods fall into two schools [3], symbolic and connectionist, although many published methods are a mixture of both.

A. Symbolic school

Symbols are used in many AI methods (e.g., states in HMMs, nodes in Graphical Models and attributes in SLAM). Although symbols are intuitive to a human programmer since he defines the associated meanings, symbols are static and have some fundamental limitations that have not received sufficient attention.

The symbolic school [4] assumes a micro-world in 4D space-time in which a set of objects or concepts, e.g., $L = \{l_1, l_2, \dots, l_n\}$, is assumed to be uniquely defined among many human programmers and their computers, represented by a series of symbols in time $\{l_1(t), l_2(t), \dots, l_n(t) \mid t_0 \leq t < t_1\}$. The correspondences among all these symbols $\{l_i\}$ of the same object across different times are known as “the frame problem” [4] in AI which means that the programmer must manually link every symbol along time with its corresponding physical object. In computer vision, the symbolic school assumes a single symbol o_i , for all its 3D positions in its 3D trajectory $\{\mathbf{x}(t) \mid t_0 \leq t \leq t_1\}$ and uses certain techniques, such as feature tracking through video (e.g., for driverless cars). Therefore, the symbolic school is based on human-handcrafted set of symbols and their assumed meanings. Marvin Minsky wrote that symbols are “neat” [5], but in fact, symbols are “neat” mainly in a single human programmer’s understanding but not between different programmers and not in relating computer programs to a real world.

We will see the Developmental Network (DN) model of a brain is free from any symbols in its full version. Abstract symbols correspond to action/state vectors in the motor area of DN. Therefore, the frame problem is automatically solved through emergent action/state vectors in a physically grounded DN, without using any symbols in the DN’s internal representations.

A major problem for symbolic AI is the generalization issue of symbols as defined here.

Definition 1 (Brittleness of static symbols): Suppose a symbolic AI machine $M(L)$ designed for a handcrafted set L of symbols is applied to a real world that requires a new set of symbols L' , with $L \cap L' \neq \emptyset$, $M(L)$ fails without a human programmer who handcrafts an appropriate mapping function $f : L' \mapsto L$ that maps every element of L' to an element in L so that $M(f(L'))$ works correctly as before.

Many expert systems (e.g., CYC, WordNet and EDR [6]) and “big data” projects [7] require a human programmer to be in the loop of handcrafting such a mapping f during deployment. For example, a machine M developed in Florida is deployed in Michigan but Michigan has snow but Florida does not. Because it is extremely challenging for a human programmer to understand many implicit limitations of $M(L)$, the mapping f that the human handcrafts typically makes $M(f(L'))$ fail, resulting in the well-known high brittleness of symbolic systems.

Due to emergent representations as numeric vectors, a DN robot discussed below learns snow settings and the snow concept when it sees snow scenes for the first time, because there are no symbols that correspond to snow in DN’s representations.

In general, the developmental methods to be discussed below automatically address such new concept problems without a need for a human programmer to be in the loop of handcrafting a symbolic mapping f during a deployment. In this paper, the author will further argue that the symbolic school suffers from human PSUTS.

B. Connectionist School

The connectionist school claimed to be less brittle [8], [9]. However, a network is egocentric—meaning that the agent starts from its own (neural) network, instead of a symbolic world. It must learn from the external *world* without a handcrafted, *world-centered* object model. Although connectionist methods often assume some task-specific symbols, e.g., a static set L of object labels, they also assume a restricted world implicitly. Therefore, a connectionist model typically needs to sense and learn from a restricted world using a network. The use of L by any neural networks (e.g., ImageNet [10] and many other competitions) as a set of object labels is a fundamental limitation that also causes the resulting system to be brittle for the same reason as the symbolic school.

Typically, a neural network is meant to establish only a mapping f from the space of input X to the space of class labels L ,

$$f : X \mapsto L \quad (1)$$

[11], [12]. X many contains a few time frames. Many video analysis problems, speech recognition problems, and computer game-play problems are also converted into this static input space so that the input space also includes L , so as to learn

$$f : X \times L \mapsto L. \quad (2)$$

Without a pressure of performance characterization during learning other than the performance of the final network, a self-organization map (e.g., SOM) has been used often as an unsupervised but slow learning method [13], [14], [15].

In contrast, with a pressure of performance characterization during learning, Cresceptron [16] used a “skull-closed” incremental-learning Hebbian-like scheme with receptive-field based competitions.

Other than the Hebbian mechanisms which are strictly “unsupervised” used by he Cresceptron and the DN explained below, two other types of learning schemes have been published:

- A Human handpicking features: after knowing the test set, humans handpick features, reported explicitly [17], [18], [19] or implicitly as “weakly supervised” [19]. This author called them “skull-open” [20].
- B Error-backprop: Locally train multiple networks each from a different set of random weights. After the training, post-select the luckiest network. Report the luckiest network only [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38] but many publications do not report the post-selection stage at al, with few exceptions [27].

Below, this author will argue that (A) suffers from human Post-Selections and (B) suffers from machine Post-Selections. While Cresceptron, the first deep learning for a 3D world, generates a single large network, it showed an impressive generalization power due to the use of the nearest neighbor scheme at every layer of an automatically generated deep network. This author will argue that Post-Selections in (A) and (B) suffer from weak generalizations (due to three types of lucks to be discussed below) and did not count the cost of training multiple networks many of which were not reported.

By the way, *genetic algorithms* offer another approach to such network learning. These algorithms study changes in genomes across different life generations. However, many genetic algorithms do not deal with lifetime development [39], [40]. We argue that handcrafting functions of a genome as a Developmental Program (DP) seems to be a clean and tractable problem, which avoids the extremely high, cost of evolution on DP. Many genetic algorithms further suffer from the PSUTS problems, since they often use test sets as training sets (i.e., vanished tests) as explained below.

Marvin Minsky [5] among many scholars complained that neural networks are “scruffy” or “black boxes”. This problem is not addressed holistically until the framework of Emergent Turing Machine was introduced [41] into Developmental Networks (DNs) by the Developmental School discussed below. A lack of Emergent Turing Machine mechanisms or being “scruffy” in sample fitting appears to be the main cause of PSUTS in traditional neural networks trained by human feature-handpicking or error-backprop methods.

C. Developmental School

The main thrust of the Developmental School, formally presented 2001 by Weng and six co-authors [39] is the task-nonspecificity for lifetime development, known as Developmental Programs (DPs) that simulate the functions of genome without simulating the genome encoding. Although a DP generates a neural network, a DP is very different from a conventional neural network in the evaluation of performance across each life—all errors from the inception time 0 of each life is recorded and reported up to each frame time $t > 0$, as explained further below.

The first developmental program seems to be the Cresceptron by Weng et al. [42], [43], [16] which appears to be, as far as the author is aware, the first deep-learning Convolutional Neural Network (CNN) for a 3D world. As explained in [44], [45] other well-known CNNs for 3D recognition, although they do not use a generative DP, followed many key ideas of Cresceptron. Cresceptron seems to be the first incremental neural network whose evaluation of performance is across its entire “life” and only one network was generated (developed) from the given training data set.

Cresceptron did not deal with time. A developmental approach that deals with both space and time in a unfired fashion using a neural network started from Developmental Networks (DNs) [46] whose experimental embodiments range from Where-What Networks 1 (WWN-1) [47] to WWN-9 [48]. The DNs went beyond vision problems to attack general AI problems including vision, audition, and natural language acquisition as emergent Turing machines [41].

DNs overcame the limitations of the framewise mapping in Eq. (2) by dealing with lifetime mapping:

$$f : X(t-1) \times Z(t-1) \mapsto Z(t), t = 1, 2, \dots \quad (3)$$

where $X(t)$ and $Z(t)$ are the sensory input space and motor space, respectively, and \times denotes the Cartesian product of sets. A fundamental difference between Eq. (2) and Eq. (3) is that in the latter the Z space contains exclusively emergent vectors, instead of any symbols, so that the actions/states are incrementally taught and learned across a lifetime.

As we will see in Section IV-A, all the errors occurred during any time of each life is recorded and taken into account in the performance evaluation.

It is important to extend Eq. (3) to include the hidden area Y that generates international (hidden) representations. To model how Y -to- Y connections enable something similar to higher and dynamic order in Markov models (but not symbolic), the above lifetime mapping is extended to:

$$f : X(t-1) \times Y(t-1) \times Z(t-1) \mapsto Y(t) \times Z(t), t = 1, 2, \dots \quad (4)$$

Note that $Z(t-1)$ here is extremely important since it corresponds to the state of a Turing machine. Namely, all the errors occurred during any time of each life is recorded and taken into account in the performance evaluation.

Different from the static symbols in the symbolic school and the space of class labels L of static symbols in Eq. (2) of the connectionist school, the space $Z(t)$ of numeric vectors of the developmental school is free from symbols. Therefore, these states/actions are directly teachable or self-generative, inspired by brains [49], [50], [51], [52], [53], [51], [54]. This new symbol-free formulation is necessary to model not only brain’s spatial processing [55] and temporal processing [56], but also Autonomous Programming for General Purposes (APFGP) [57]. Based on the APFGP capability, we open the door towards the next step—conscious learning [58]—learning while being partially and increasingly conscious. By *conscious learning*, we do not mean “open-skulledly” handcrafting general-purpose consciousness, which is probably too complicated to handcraft. But instead we enable fully autonomous machine learning while machines being partially conscious—autonomously learn more sophisticated consciousness skills using their partial, earlier, and simpler conscious skills across the lifetime.

This is the journal archival version of the earlier conference papers [59], [60] with significantly refined additional material and analysis.

In the following, we will discuss Post-Selection in Section II. Section III addresses why error-backprop algorithms suffer from severe local minima problems. Section IV explains how a Developmental Network

solves the local minima problems, since only one network is needed for each life and the evaluation of performance across the entire life. Section V discusses experiments. Section VI provides concluding remarks.

II. POST-SELECTIONS

AI has made impressive progress, gained much visibility, and attracted the attention of many government officials. However, there are protocol flaws that have resulted in misleading results.

First, let us consider three learning conditions that any fair comparisons of AI methods should take into account.

A. The Three Learning Conditions

Many AI methods were evaluated without considering how much computational resources are necessary for the development of a reported system. Thus, comparisons about the performance of the system have been tilted toward competitions about how much resources a group has at its disposal, regardless how many networks have been trained and discarded, and how much time the training takes.

Here we explicitly define the Three Learning Conditions for development of an AI system:

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

The competing standard of the ImageNet competitions [61] did not include any of these three conditions. The AIML Contests [62] considered all the three in performance evaluation. In the following Subsection, we discuss why task-nonspecificity and incremental mode should be considered in any comparisons.

B. Task-specific vs. Task-nonspecific

A task-specific learning approach learns less because much is handcrafted by a human according to the given task. Furthermore, a task-specific method is brittle.

In Condition (1) of the Three Learning Conditions, the task-nonspecific learning paradigm is significantly different from the task-specific traditional AI paradigm as explained in Weng et al. 2001 [39]. In a task-specific paradigm, the system developer is given a task e.g., constructing a driverless car. Then, it is a human programmer who chooses a world model, such as a model of lane edges. Next, he picks an algorithm based on this world model, e.g., the well-known Hough transform algorithm [63], [64] for line detection which makes every pixel that is detected as edge cast votes for lines of all possible orientations o and distances d from the origin that go through the pixel. Then the top-two “peaks” of line parameters (o, d) that have received the highest votes are adopted to declare a line detected from the image. Here “edges” and “lane lines” are two symbolic concepts picked up by the programmer. Such systems will fail when lanes are unclear or totally disappear due to weather or road conditions, leading to a brittle system. Human brains appear to be more resilient.

In contrast, a task-nonspecific approach [39] not only avoids any symbolic model, but also does not require that a task is given. The desirable actions at any time are taught, tried, and recalled automatically by the learner based on system’s learned context q [65] that includes automatically figured-out goal and state, as well as the current input. The mapping function $f(\mathbf{z}, \mathbf{x}) = \mathbf{z}'$, representing the symbolic mapping $f_s(q, \sigma) = q'$, corresponds to a finite automaton. Weng has proven that the control of any Turing machine is a finite automaton [41]. Thus, this framework is of general-purposes in the sense of universal Turing machines. Any universal Turing machine is of general purposes, because it can read any program written for any purposes and run it for the purposes. Any neural network that learns a universal Turing machine becomes of general purposes in the sense of any programs, not just in the sense of any mappings like that in Eq. (1). Thanks to the absence of any world model, such as lanes, this task-nonspecific approach has a

potential to be more robust than a world-model based approach. The task-nonspecific approach typically uses a neural network to learn because the need to learn vector mapping function $f(\mathbf{z}, \mathbf{x}) = \mathbf{z}'$. We will discuss internal response vector \mathbf{y} in Section IV but task-nonspecificity holds true without \mathbf{y} .

C. Batch vs. Incremental Learning Modes

Neural network learning for the mapping f has two learning modes, batch learning and incremental learning.

With batch learning, a human first collects a set D of data (e.g., images) and then labels each datum with a desirable output (e.g., command of navigation or class label). A neural network is trained to approximate a mapping f in Eq. (1) or Eq. (2). Many batch-learning projects use an error-backprop method [23], [66], [24] which uses a gradient-based method to find a local minimum in error.

As we will discuss in Section III, the gradient in the error-backprop method does not contain key information of many other data if the learning mode is incremental. Thus, error-backprop on a large data set does poorly using a purely incremental learning mode. Many used a block-incremental learning mode which suffers from the big data flaw in Theorem 1 below.

In contrast, all developmental methods cited here use incremental learning mode for long lifetimes, using a closed-form solution to the global lifetime optimization. The competition among neurons guarantees that the winner is the most appropriate neuron whose memory corresponds to the current working memory [67].

However, the batch and incremental learning modes are not capability-equivalent [67]. The former requires all sensory inputs are available at a batch, independent of the corresponding actions. Therefore, the former is easier and also incorrect according to sensorimotor recurrence. By sensorimotor recurrence, we mean that sensory inputs and motor outputs are mutually dependent on each other in such a recurrent way that off-line collection of inputs are technically flawed. We have the following theorem:

Theorem 1 (Big Data Flaw): All static “big data” sets used by machine learning violate the *sensorimotor recurrence* property of the real world.

Proof: A learning agent at time $t - 1$, as shown in Eqs. (3) and (4) does not have the next sensory input from $X(t)$ available before the corresponding actions in $Z(t - 1)$ are generated and output, since the sensory input in $X(t)$ varies according to the agent actions in $Z(t - 1)$. As an example, turning head left or right will result in a different image sensed. Therefore, all static “big data” sets violate the sensorimotor recurrence. ■

One may say that classifications of static images are fine. We do not agree, because even when a human (or machine) is looking at a static image, he uses attention (e.g., context-based saccades) which is a sequence of actions. Each saccade results in a different fovea image.

Therefore, incremental learning is necessary for the *sensorimotor recurrence*. All batch-training methods use a static set of training data and, therefore, are inappropriate for any of them to claim near-human performance since the two learning problems are different. This leads to the following theorem.

Consider a hierarchy of levels of object types, such as nails, fingers, palms, hands, arms, limbs, torsos, human bodies, etc. Because vision requires a high level l to understand natural scenes with abstraction of parts with invariances (e.g., all fingers of different scales, looks, and at different locations), each child needs an open-ended world to learn to learn rules (e.g., finger-parts and hand-whole) instead of simple-minded pattern recognition of sensory images.

Theorem 2 (Nonscalability of Big Data without abstraction): All static “big data” sets used by machine learning are nonscalable if they are treated as pattern recognition without rule abstractions.

Proof: Suppose that a static data set D has shown the presence of k feature types defined at level 1 (e.g., edge pixels are a type). Suppose a combination of $k > 1$ feature types to level $l+1$ type (e.g., straight line type is from multiple edge pixels) is defined from k types of feature types at level l , $l = 1, 2, \dots$. The number of samples for a l -level feature type requires at least k^l observations to discover all necessary within-type equivalence (e.g., logic OR is at $l = 2$ with $k = 2$ logic features at $l = 1$, thus without rule

abstraction (e.g., parts and whole), it requires $k^l = 2^2 = 4$ observations, corresponding to 4 rows in the truth table of logic OR). Since $f(l) = k^l$ is an exponential function in l , k^l quickly exceeds any fixed number of observations in the static data set D . ■

Rule abstractions deal with invariances. For example, a “what” concept is “where”-invariant and a “where” concept is “what”-invariant, as explained in [55], [68].

Section IV discusses an optimal framework through which such abstractions can take place from learning simple rules during early life that enable learning of more complex rules during later life—called scaffolding [69].

Theorem 2 leads to two observations on **data fitting on a static data set**:

Observation 1: Any **data fitting on a static data set** without learning invariant concepts are nonscalable, including the n -fold cross-validation discussed below. Unfortunately, **data fitting on a static data set** is a norm in all ImageNet Contests [66]. Namely, the remaining subsections in this section analyze approaches that are nonscalable. For example, computer vision is not a “one-shot” pattern classification problem as argued by Li Fei-Fei et al. [19] (which was questioned in PubMed without responses), but rather a spatiotemporal problem to learn various invariant concepts present in cluttered natural scenes through autonomous attention saccades, as explained further in Observation 2.

Observation 2: Learning invariant concepts seem nonscalable for any **data fitting on a static data set** either, because there are too many images to be labeled by hand (e.g., all pixel locations) [55], [68]. Like a human baby, any scalable machine learning methods must be conscious through which the machine learner must consciously guess concepts (i.e., not just active learning [70]) (e.g., an object type) and verify their invariance rules (e.g., the where-invariance of a what concept). The state-based transfer in Theorem 8 of [56] explains how each concept state reduces the number of samples to be learned from an exponential k^l down to only kl (see Fig. 6 of [56] for intuition where $k = 10$ and $l = 3$). Thus, Section IV not only addresses the non-scalability problems in this section, but is also necessary for conscious learning whose theory was recently published in Weng 2020 [58] with some single-sensory-modality experimental results, but animal-level conscious robots that are multi-sensory and multi-motor have not yet been demonstrated. The availability of real-time learning brain-chip is a current bottleneck.

D. Fitting, validation and test errors

Given an available data set D , D is divided by a partition $P = (T, V, T')$ into three mutually disjoint sets, a training set T , a validation set V , and a test set T' so that

$$D = T \cup V \cup T'. \quad (5)$$

Two sets are disjoint if they do not share any elements. The validation set is possessed by the trainer, the test set should not be possessed by the trainer since the test should be conducted by an independent agency. Otherwise, V and T' become equivalent.

As we will see in Section III, given any architecture parameter vector \mathbf{a}_i , it is unlikely that a single network initialized by a set of random weight vectors can result in an acceptable error rate on the training set, called fitting error, that the error-backprop training intends to minimize locally. That is how the multiple sets of random weight vectors come in. For k architecture vectors \mathbf{a}_i , $i = 1, 2, \dots, k$ and n sets of random initial weight vectors \mathbf{w}_j , the error back-prop training results in kn networks

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

Error-backprop locally and numerically minimizes the fitting error $f_{i,j}$ on the training set T .

[27] seems to have mentioned $n = 20$. [71] did not give n but seems to have mentioned 60 million parameters which probably means each \mathbf{w}_i and each \mathbf{a}_j combined to be of 60 million dimensional. Using the above example of $k = 3^{10} = 59049$, $kn \approx 1M$ networks must be trained, a huge number that requires a lot of computational resources to do number crunching and a lot of manpower to manually tune the range of hyper-parameters.

Definition 3 (Distribution of fitting, validation and test errors): The distributions of all kn trained networks' fitting errors $\{f_{ij}\}$, validation errors $\{e_{ij}\}$, and test errors $\{e'_{ij}\}$, $i = 1, 2, \dots, k$, $j = 1, 2, \dots, n$ are random distributions depending on a specific data set D and its partition $P = (T, V, T')$. The difference between a validation error and a test error is that the former is computed from the same group using a group-possessed validation set V but the latter is computed by an independent agency using a group-unknown test set T' .

We define a simple system that is easy to understand for our discussion to follow.

Definition 4 (Nearest neighbor classifiers with a confidence threshold): Define a network stores the entire training set T . Suppose the input x matches the nearest sample s in T . If the distance between x and s is not larger than a confidence threshold d (a hyper-parameter), then the network outputs the associated label of the nearest sample s . Otherwise, the system outputs "unknown".

Namely, this system uses a lot of resources for over-fitting. It gives up if the distance is larger than d , but has a perfect fitting error (zero) for any positive d .

A neural network architecture has a set of hyper parameters represented by a vector \mathbf{a} , where each component corresponds a scalar 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, $A = \{\mathbf{a}_i \mid i = 1, 2, \dots, k\}$. Suppose there are 10 scalar parameters in each vector \mathbf{a}_i . For each scalar parameter x of the 10 hyper parameters, we need to validate the sensitivity of the system error to x . With uncertainty of x , we estimate its initial value as the mean \bar{x} , positively perturbed estimate $\bar{x} + \sigma_x$ (σ is the estimated standard deviation of x), and negatively perturbed estimate $\bar{x} - \sigma_x$. If each scalar hyper parameter has three values to try in this way, there are a total of $k = 3^{10} = 59049$ architecture parameter vectors to try, a very large number. For example, the initial threshold \bar{d} in the nearest neighbor classifier can be estimated by the average of nearest distance between a sample in V and the nearest neighbor in T and the σ_d be estimated by the standard deviation of these nearest distances.

Let us define the Post-Selection. Suppose that the trainer is first aware of the validation sets (or the test sets).

Definition 5 (Post selection): A human programmer trains multiple systems using the training set T . 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 set V (or the test set T'). This is called Post-Selection—selection of one network from multiple trained and verified (or tested) networks.

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

First, consider Post-Selection Using Validation Sets (PSUVS):

E. 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 luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches *the error of the luckiest-architecture and the luckiest initial weights from Post-Selection on Validation Sets*:

$$e_{i^*,j^*} = \min_{1 \leq i \leq k} \min_{1 \leq j \leq n} e_{i,j} \quad (6)$$

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

Similarly, a human PSUVS is a procedure wherein a human selects a system from multiple trained systems for $\{e_{i,j}\}$ using human visual inspection of internal representations of the system and their validation errors.

F. Cross-Validation

The above PSUVS is an absence of cross-validation [72]. Originally, the cross-validation is meant to mitigate an unfair luck in a partition of the dataset D into a training set T and a test set T' (empty validation set). For example, an unfair luck is such that every point in the test set T' is well surrounded by points in the training set T . But such a luck is hardly true in reality.

To reduce the bias of such a luck, an n -fold cross-validation protocol, $n \geq 2$, divides the data set D into n subsets of same size and conducts n experiments. The term “cross” refers to switching the roles of training and testing data. In the i -th experiment, the i -th subset is left out as the test set and the remaining $n - 1$ folds of data form the training set. Thus, the cross-validation protocol conducts n experiments, for $i = 1, \dots, n$, to obtain n errors, e_1, e_2, \dots, e_n . The *cross-validated error* is defined as the average of errors from the n tests, to filter out the partition lucks:

$$\bar{e} = \frac{1}{n} \sum_{i=1}^n e_i \quad (7)$$

as well as the distribution of errors $\{e_j\}$.

The n different numbers here shows a distribution show a distribution $\{e_j\}$ to indicate how sensitive the error is to lucks, such as the number of partition pairs between a training set and a validation/test set, the number of tried random seeds for initializing network weights, the number of tried hyper-parameter vectors, or a combination thereof. The larger the n , the better the estimated standard deviation of $\{e_j\}$.

G. Types of lucks in a Neural Network

In a neural network, there are at least three kinds of lucks:

Type-1 order lucks: The luck in a partition P_i into a training set T_i and a test set T'_i from a data set D resulting in test error e_i , $i = 1, 2, \dots, n$. Different partitions correspond to different luck outcomes. This kind of outcome variation results in a variation of performance from different outcomes. Conventionally, this type of lucks is filtered out by cross-validation (e.g., n -fold cross-validation) as well as reporting the deviation of $\{e_i\}$ during the cross-validation. However, such cross-validation and deviation have hardly published for neural networks and reported. The smaller the average \bar{e} of $\{e_i\}$, the more accurate the trained network is; the smaller the standard deviation of $\{e_i\}$, the more trustable the average error \bar{e} is.

Type-2 weights lucks: As we will discuss below, weights specify the role assignment for all the neurons in the neural network. A random seed value determines the initialization of a pseudo-random number generator, which gives initial weights \mathbf{w}_i for a neural network $N(\mathbf{w}_i)$, resulting in a test error e_i , $i = 1, 2, \dots, n$, after training of these n networks and testing on T' . It is unknown that such a luck will be carried over to a new test set T'' that is outside the data set D but was drawn from the same distribution of S . Because a neural network might not capture the internal rules of the training set T , this paper argues that a statistical validation of the reported error should be performed by reporting the distribution of $\{e_i | i = 1, 2, \dots, n\}$, where e_i is from a different initial weight vector \mathbf{w}_i . For example, Krizhevsky et al. [71] reported 60 million parameters, mostly in \mathbf{w}_i but only the luckiest e_i was reported. The smaller the average \bar{e} of $\{e_i\}$, the more accurate the trained network is; the smaller the standard deviation σ of $\{e_i\}$, the less sensitive the trained neural network is to the initial weights and thus the accuracy is more trustable for real applications. For i.i.d. (identically independently distributed) errors, we can expect that doubling the number n will reduce the expected variance of \bar{e} by a factor $1/\sqrt{2}$, since the expected variance of n random numbers is about σ^2/n .

Type-3 architecture lucks: The initial hyper-parameter vector \mathbf{a}_j of the neural network gives an error e_j , $j = 1, 2, \dots, k$. Because such a luck of \mathbf{a}_j might not capture the internal rules of the training set T_j , this paper argues that a statistical validation of the reported error estimate should be performed and the distribution of $\{e_j\}$ be reported. In our above example, the number of distinct hyper-parameter vectors to be tried is $k = 3^{10} = 59049$. The smaller the average \bar{e} of $\{e_j\}$, the more accurate the trained network

is; the smaller the sample variance of $\{e_j\}$, the more trustable \bar{e} is, namely, the average error \bar{e} is less sensitive to the initial hyper-parameters of the network. For example, the threshold d of the nearest neighbor classifier in Definition 4 might result in a large deviation. A good way is to reduce the manual selection nature of such hyper-parameters. For example, all hyper-parameters are adaptively adjusted from the initial hyper-parameters that are further automatically computed from system resources, e.g., the resolution of a camera, the total number of available neurons, and the firing age of each neuron [67].

For notation clarity in the discussion that follows, index j is used in Type 3 to distinguish index i in type 2, but the above three types of lucks are all different.

Let us discuss the case of a developmental network, such as Cresceptron [16] and DN [41]. Type-1 cross-validation is not needed because of reporting of a lifetime error. In other words, errors of all new tests in each life are taken into account throughout the lifetime. Type-2 validation is not needed because all different random weights w_i leads to the function-equivalent neural network under certain conditions. For example, in top- k competition, with $k = 1$ different w_i give the exactly the same neural network and with $k > 1$ different w_i give almost the same neural network. The distribution of lifetime errors $\{e_i\}$ is expected to have a negligible deviation across different initial weight vectors w_i , given the same Three Learning Conditions. Type-3 validation might be useful but is expected to be negligible since the most obvious parameters such as learning rate and momentum of learning rate is automatically and optimally determined by each neuron, not handcrafted, as in LCA [67]. The synaptic maintenance automatically adjusts all receptive fields [73], [74] so that the neural network performance is not sensitive to the initial hyper-parameters.

In contrast, a batch-trained neural network typically uses a Post-Selection to pick the luckiest network without cross-validation for either of the above three types of lucks, e.g., in ImageNet Contest [61]. Namely, errors occurred during batch training of the network before the network is finalized and how long the training takes are not reported. Below, Fig. 5 will show a huge difference between the luckiest CNN with error-backprop and the optimal DN. Many researchers have claimed error-backprop works without providing much-needed three types of validations.

Next, let us discuss Types 2 and 3 validations which are new for neural networks but hardly done.

H. Post-Selection with Types 2 and 3 Average-Validations

Type-1 cross-validation should be nested inside the Types 2 and 3 validations, but this triple-nested protocol could be too computationally expensive. Below, we delay Type-1 cross-validation till after Type 2 and Type 3 validations.

Assume that we use n random weight vectors w_i and k grid-search hyper parameters a_j . Each combination of w_i and a_j gives an error $e_{i,j}$ from the corresponding validation set. To reduce the effect of such a luck for each vector w_i , an average of $e_{i,j}$ over n values of i should be used instead of the minimum in Eq. (6). This leads to *the random-weights validated error for the luckiest architecture* from PSUVS:

$$\mathbf{a}^* = \arg \min_{1 \leq j \leq k} \frac{1}{n} \sum_{i=1}^n e_{i,j}. \quad (8)$$

We dropped the term “cross” because this validation examines other random seeds without switching the roles between training and testing.

Similarly, we define *the hyper-parameter validated luckiest initial weights* from PSUVS:

$$\mathbf{w}^* = \arg \min_{1 \leq i \leq n} \frac{1}{k} \sum_{j=1}^k e_{i,j}. \quad (9)$$

We dropped the term “cross” for the same reason.

From a statistical point of view, the initial hyper parameter vector \mathbf{a}^* and the random initial weights \mathbf{w}^* validated above through averages should be more robust in real applications than those without average-validation in Eq. (6).

For both the luckiest \mathbf{a}^* and \mathbf{w}^* , the standard deviation under min should be reported to show how sensitive the reported performance is to the validation process. If the variation is large, the corresponding network is not very trustable in practice.

We also need to be aware of another protocol flaw: Random seeds and hyper parameters are all coupled. Under such a coupling, Type 2 validation seems unnecessary with $n = 1$ but the search of the luckiest weights is embedded into the search for the luckiest hyper-parameter vector where each hyper parameter vector uses a different seed. Similarly, Type-3 validation seems unnecessary with $k = 1$ but the search of the luckiest hyper-parameter vector is embedded into the search for the luckiest weights, where each random seed uses a different hyper parameter vector.

Since a PSUUVS 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 a validation set does not guarantee doing well on a test set. Typically, due to a very large number of samples, availability of validation sets and unavailability of test sets in a properly managed contest, principles of Post-Selection should cause the validation error rate to be smaller than the test error rate. (However, in Table 2 of [71], the test error rate is smaller than the validation error for 7CNNs, causing a reasonable suspicion that PSUTS could be used instead of PSUUVS.)

The following subsection discusses the luckiest network with the luckiest hyper-parameter vector \mathbf{a}^* and the luckiest initial weights \mathbf{w}^* .

I. The Luckiest Network from a Validation Set

Many people may ask: Are there any technical flaws in at least PSUUVS, since it does not use the test sets? We analyze the luckiest network in this section and reach a conclusion that any post-selection is technically flawed and results in misleading results, including both PSUUVS and PSUTS. However, in general, Type-1 cross-validation is to filter out lucks in data partition that a typical user does not have during a deployment of the method. Namely, it is a severe technical and protocol flaw in reporting only the luckiest network, regardless the post-selection uses validation sets or test sets.

This conclusion has a great impact on evolutional methods that often report only the luckiest network, instead of those of all networks in a population. Namely, the performances of all individual networks in an evolutionary generation should be reported.

For simplicity, we assume that the space S , from which random samples in D are drawn, is static. Our conclusions here can be readily extended to a time varying D but the technical flaws are even worse.

From the sample space S , randomly draw a data set D . D is partitioned into three mutually disjoint sets, training set T , validation set V and test set T' , so that Eq. (5) holds true. For realistic applications, we should assume that T , V and T' are mutually independently drawn from S so that T , V and T' are mutually independent. Identically independently distributed (i.i.d.) is a sufficient condition, but we do not need such a restrictive condition because temporal-dependency often occurs in lifetime development. Namely, we only need that any three vectors from T , V and T' , respectively, are mutually independent.

Using the training set T , one trains kn networks, where k and n are the number of hyper-parameter vectors \mathbf{a} 's and random weight vectors \mathbf{w} 's, using a training algorithm (e.g., error-backprop),

$$N(\mathbf{a}_i, \mathbf{w}_j) \leftarrow f_{\mathbf{a}_i, \mathbf{w}_j}(T). \quad (10)$$

This is like a teacher trains kn students in a class. The teacher knows that the fitting error on T does not predict the validation error well, due to the possibility of over fitting. One extreme example is the above nearest-neighbor classifier with confidence $d = 0$.

The teacher then tests each $N(\mathbf{a}_i, \mathbf{w}_j)$ on the validation set V to get $e_{i,j}$. This is like the teacher observes the performance of kn networks in a mock exam.

The teacher then post-selects and reports only the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ whose validation error e_{i^*, j^*} is minimum in Eq. (6). This is like the teacher colludes with the Educational Test Service (ETS) so that the ETS only reports the luckiest network but not all remaining $kn - 1$ networks to cover up.

J. Luckiest Network with Type-1 Cross-Validation

Suppose that a user has bought the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ and test on his new test data T' randomly drawn from S , independent of T and V . The luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ that reached the minimum error rate in V does not mean that it reaches the minimum error rate in T' . Because T' is independent of T and V , and $\mathbf{a}_{i^*}, \mathbf{w}_{j^*}$ are luckiest on a particular pair (T, V) only, we need to compute the expected error rate of $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ on T' .

Theorem 3 (Type-1 cross-validation of the luckiest): The luckiest network on validation set gives an error rate that is approximately the average error in Type-1 cross-validation, supposing that, in an n -fold cross validation, n folds of data are drawn i.i.d. (independently and identically distributed) among folds from data set D , but individual samples inside each fold do not need to be i.i.d.

Proof: Let F denote the event that both the training set and validation set are from a fixed data set D from S . Consider in a real application, n tests were conducted on the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ using T'_i , $i = 1, 2, \dots, n$, where each partition $P_i = (T_i, V_i, T'_i)$ in each of the i -th training and test pair is drawn from the real application space S . We compute the average error rate from the luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$:

$$\begin{aligned} e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}) &= \frac{1}{n} \sum_{i=1}^n e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i) \\ &\approx \frac{1}{n} \sum_{i=1}^n e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i | F) \end{aligned} \quad (11)$$

where the term $e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i)$ means the error of the luckiest network using training set T_i , validation set V_i , and test set T'_i , and $e_i(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}; T_i, V_i, T'_i | F)$ means the same but T_i, V_i, T'_i are all from the same D as one does in n -fold cross-validation. ■

Note, the n -fold i.i.d. is weaker than i.i.d. for all samples. In practice, i.i.d. is rarely true even for pattern recognition problems, such as image classification due to sequential attention discuss above. Also note that the left side of \approx sign in Eq. (11) is expected larger because the data T_i, V_i, T'_i on the right side are all from a fixed D but the left side does not have such a restriction.

The above theorem tells us that the error rate of the luckiest network from a single validation set in PSUVS is misleading without any partition validation. This is because the error rate is a random function, depending on not only many random initial weights, many hyper parameters, and local lucks of error-backprop, but also a particular partition (T, V, T') . This seems especially true if the data D were made public and overworked during 2010-2014 [61, p. 213].

In practice, when we report an error rate $e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ which is always a random number x , depending on how much hand tuning is done, how much computational resources are used for a large-scale search for the random seeds and hyper-parameters, as well as the validation or a lack thereof. We should also report the distribution of this random number x , such as the maximum, 75%, 50%, 25%, and the minimum value of x , over multiple training-and-test pairs in cross-validation, random seeds and hyper-parameters. Otherwise, the error rate, if only as a single number x , is misleading, since users of this learning method or buyers of the luckiest network do not have the same partition luck.

Up to now, this author has not found any published papers that report not only the luckiest network from error-backprop but also Type-1, Type-2 and Type-3 validations. Many papers do not report the post-selection stage at all [24], [25], [26], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], except [27], let alone whether the reported error is from the validation error V or the test set T' .

Next, we discuss Post-Selections Using Test Set (PSUTS). There are two kinds of PSUTS, machine PSUTS and human PSUTS.

K. Machine PSUTS

If the test set T' is available which seems to be true for almost all neural network publications other than competitions, 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 luckiest network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches the minimum, called *the error of the luckiest architecture and the luckiest initial weights from Post-Selection on Test Set*:

$$e'_{i^*, j^*} = \min_{1 \leq i \leq k} \min_{1 \leq j \leq n} e'_{i,j}. \quad (12)$$

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. (7) to give *the error of the luckiest architecture with validated weights from PSUTS*:

$$\mathbf{a}_{*,j^*} = \arg \min_{1 \leq j \leq k} \frac{1}{n} \sum_{i=1}^n e'_{i,j}. \quad (13)$$

But the above error is still flawed since each term under minimization has peeked into test sets. Instead, it is better to use Eq. (8) which does not use the test sets. Of course, the test error rate of that in Eq. (8) tends to be larger than that from Eq. (13).

A similar discussion can be made for *the error of the luckiest initial weights with validated architecture from PSUTS*. Do not peek 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.

In general, the more free parameters a network has, the more likely the network can report an artificially small error as in Eq. (12). That is why we need the computational resource in the Three Learning Conditions.

Although PSUVS has flaws of post-selection and a lack of three types of validation, the key difference between PSUVS and PSUTS does not guarantee that PSUVS reports a low error rate as PSUTS. In fact, it is expected that the luckiest network from PSUVS does better on a validation set V than on a test set T' because the Post-Selection did not “see” the test set T' but “saw” the validation set V . Likewise, it is expected that the luckiest network from PSUTS does better on the test set T' than on a validation set V because the Post-Selection did not “see” the validation set V but “saw” the test set T' . In the following paragraph, we discuss that this expectation is reversed in Table 2 of [71, page 88].

In ImageNet Contest 2012, the test sets were released to competition teams over 2.3 months ahead of the output-result submission date. Although the class labels were not attached to the test sets other than being available indirectly through an online test server provided by the contest organizers, it was not difficult to “crack” a test set by manually hand-labeling the test set. The first author of [71] seems not sensitive to the fundamental difference between a validation set and a test set by writing: “in the remainder of this paragraph, we use validation and test error rates interchangeably”. By “we cannot report test error rates for all the models that we tried” [71, page 88], there is no evidence to rule out what he meant was the possibly “cracked” test set is not necessarily exactly the same as the original test set. But in Table 2 of [71, page 88], the 7NNs did worse on the validation set (possessed) than the test set (if not “cracked” and searched for minimization like in Eq. (12)). This reversed our expectation in the previous paragraph. Is it an evidence of using PSUTS instead of PSUVS?

Another interesting phenomenon that is consistent with the likely use of PSUTS instead of PSUVS is that the SuperVision Team of ImageNet Contest 2021 did not submit any output results for “the fine-grained classification task, where algorithms would classify dog photographs into one of 120 dog breeds” [61, footnote, p214]. It appears that cracking “120 dog breeds” is harder than cracking “a list of object categories present in the image” where the class labels are all available in the provided training sets. [71] lacks due transparency about the post-selection stage except that Geoffrey Hinton admitted the “luckiest” network in his brief PubPeer response to questions raised on PubPeer towards [24].

For more examples, see Fig. 1 from [75, Fig. 7], error-backprop consistently results in lower validation accuracies than the test accuracies (about 0.5% lower compared to about 0.1% lower in [71]). Are they

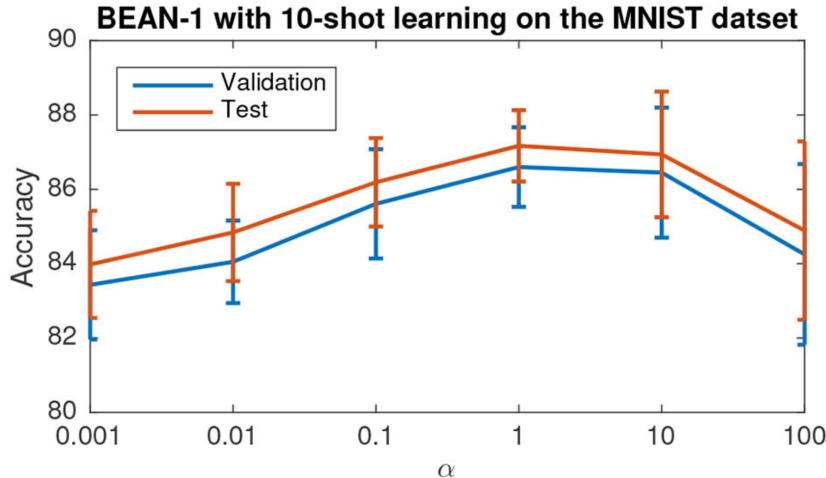


Fig. 1. The average \bar{e} and the standard deviation of $\{e_i\}$ for different values of a regularization hyper parameter α . Adapted from [75]

other evidences of using PSUTS instead of PSUVS, similar to [71]? The availability of test sets to the programmers in a project seems to be indeed addictive towards PSUTS, away from PSUVS. The standard deviation around 1% is clashes with our Theorem 4. Our experience with our own experiments with error-backprop training for CNN indicated that the maximum and the minimum values of the distribution of fitting accuracies are drastically different for different random seeds, with fitting accuracies spreading uniformly between 20% and 90%. Section III will discuss why. If Theorem 4 in Section III is correct, the deviation bars seem too small and the 20 runs in Fig. 7 of [75] could be the best 20 among many more random-seeds the programmer has tried. We hope that authors provide the source program.

L. Implications of PSUTS

Although the set $\{e_{i,j} | i = 1, 2, \dots, n; j = 1, 2, \dots, k\}$ is large, it is necessary to present some key statistical characteristics of its distribution. For example, rank all errors in decreasing order, for each type of errors, fitting, validation and test. Then give the maximum, 75% (in ranked population), 50% (median), 25%, the minimum value, and the standard deviation of these kn values for the fitting errors, validation errors, and test errors, respectively, not just the standard deviation in Fig. 1. Such more complete information of the distribution is critical for the research community to see whether error-backprop can indeed avoid local minima in deep learning as some authors claimed. Furthermore, such information is also important for the authors to show that the luckiest hyper-parameter vector is not just an over fitting to the validation/test set. Unfortunately, none of [23], [66], [24], [25], [27], [76], [29], [31] reported such distribution characteristics other than the minimum value e'_{i^*,j^*} .

Furthermore, such a use of 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 grading. This practice could hardly be acceptable to any test agencies and any agencies that will use the test scores for admission purpose since this submitted error e'_{i^*,j^*} misleads due to its lack of validation.

The error-backprop training tends to locally fit each network on the training set T ; while the Post-Selection picks the luckiest network with parameter vector \mathbf{a}_{i^*} and initial weights \mathbf{w}_{j^*} that has the best luck on T' . If an unobserved data set T'' , disjoint with T' , $T' \cap T'' = \emptyset$, is observed from the same distribution S , the error rate e''_{i^*,j^*} of $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ is predicted to be significantly higher than e'_{i^*,j^*} ,

$$e''_{i^*,j^*} \gg e'_{i^*,j^*} \quad (14)$$

because Eq. (12) depends on the test set T' in the post selection from many networks. Of course, handling a new test is also challenging for a human student but a human learning involves learning invariant rules. Any PSUTS is a technically flawed protocol.

PSUTS is tempting especially when test sets are available to the authors of paper. During all error-backprop related paper reviews I have not yet found a case in which the authors did not admit that they used USUTS when I asked. The second author of [27] claimed to have used PSUVS through a personal email to me but the first author who probably performed the experiments did not claim the same. No authors of [27] responded to PubPeer questions towards [27].

Weng 2020 [77], [78] argued that the claims by some public speakers that such misleading errors have approached or even succeeded human performance [61] are controversial, since there are no explicit competition rules that ban test sets to be used for Post-Selections.

M. Human PSUTS

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

Definition 6 (Human PSUTS): After planning experiments 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 and collect the test set T' . The key to an acceptable test score lies in how much detail the human designer can plan for what is in the test sets and how much freedom s programmer has in hand picking features.

Poggio et al. [79] and Fukushima et al. [17] explicitly admitted their use of human PSUTS. Li Fei-Fei at al. [19] only vaguely admitted their use of human PSUTS by a vague term “weakly supervised” using an extension of formulation by Pietro Perona that is originally unsupervised. Questions raised towards [19] on PubPeer were not answered by the authors.

III. WHY ERROR-BACKPROP NEEDS PSUTS

This section discusses a global view, which is new as far as the author is aware, about why error-backprop suffers from local minima even in the easier batch-learning mode.

Since error-backprop does not perform well for incremental learning mode as we can see why also from the following discussion, we will concentrate on batch learning mode. Namely, we let the network “see” the entire training set T for each network update.

Let us first consider a well-known neuronal model that is applicable to many CNNs. Suppose a post-synaptic neuron with activation z_j is connected to its pre-synaptic neurons y_i , $i = 1, 2, \dots, n$, through synaptic weights w_{ij} , by the expression:

$$\phi\left(\sum_{i=1}^n w_{ij}y_i\right) = z_j \quad (15)$$

where $\phi(y) = \frac{1}{1+e^{-y}}$ is the logistic function. The gradient of z_j with respect to weight vector $\mathbf{w}_j = (w_{1,j}, w_{2,j}, \dots, w_{n,j})$ is

$$\eta(y_1, y_2, \dots, y_n) \triangleq \eta\mathbf{y}$$

where η is the partial derivative of $\phi(y)$. Thus, according to gradient direction, the change of the weight vector \mathbf{w}_j is along the direction of pre-synaptic input vector \mathbf{y} . If the error is negative, z_j should increase. Then the weight vector should be incremented by

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + w_2\mathbf{y} \quad (16)$$

where w_2 is the learning rate. We use the w_2 to relate better the optimal Hebbian learning, called LCA, used by DN in Section IV. At this point, the following theorem is in order.

Theorem 4 (Lacks of error-backprop): Error-backprop lacks (1) energy conservation, (2) an age-dependent learning rate, and (3) competition based role-determination.

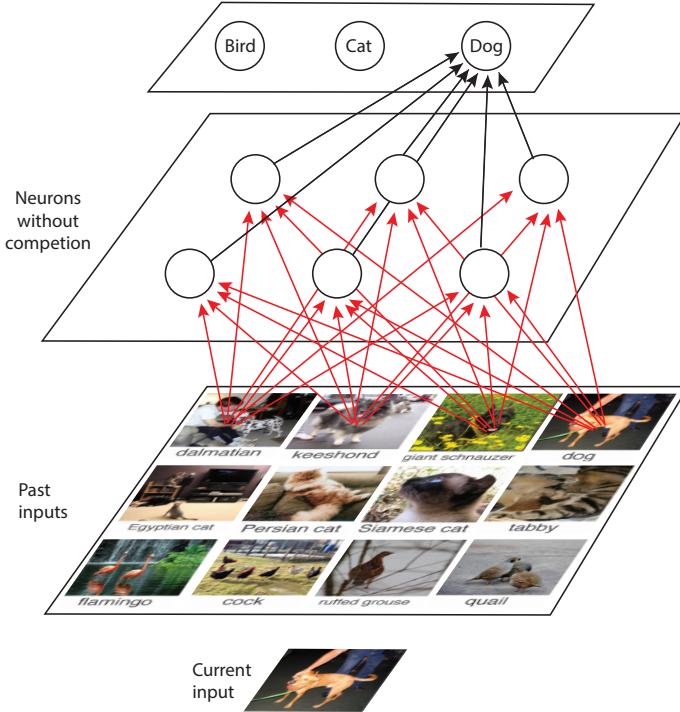


Fig. 2. Lack of role-determination in hidden neurons due to a lack of competition. The same ideas are true for a deeper hierarchy. Color sample images courtesy of [61].

Proof: Proof of (1): If pre-synaptic input vectors $\{y\}$ are similar, multiple applications of Eq. (16) add many terms of $\{w_2y\}$ into the weight vector w_j causing it to explode, which means a lack of energy conservation. Proof of (2): w_2 is typically tuned by an *ad hoc* way, such as a handpicked small value turned by a term called momentum, instead of being automatically determined in Maximum Likelihood optimality (ML-optimality) by neuronal firing age to be discussed in Section IV. Proof of (3): Suppose neuron z_j is in a hidden area of the network hierarchy. This neuron z_j updates its pre-synaptic weight using Eq. (16) regardless z_j is role-responsible or not for the current network error. Likewise, looking upstream, there is also a lack of role-determination in the gradient-based update for pre-synaptic neurons y_1, y_2, \dots, y_n , all of which must update their own weights using their own gradients. Namely, there is no competition-based role-determination in error-backprop . ■

The meaning (3) of Theorem 4 are illustrated by Fig. 2. CNNs do not have a competition mechanism in any 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 characterized by its label “neurons without competition”. The hidden neurons in this area do not have a competition mechanism which would, like in LCA [67], allow a small proportion of neurons to win the competition and fire so that they automatically take the roles that they happened to compete well. This analysis leads us to the following theorem.

Theorem 5 (Random roles in error-backprop): A set of random initial weights in a network assigns random roles to all hidden neurons, from which a local minimal point based on error-backprop learning inherits this particular random-role assignment. Which neurons in each hidden area take a role does not matter, but how hidden neurons share a set of roles in each hidden area does matter in the final fitting error, validation error, and test error after error-backprop.

Proof: Without loss of generality, suppose a maximum in the output neuron means a positive

classification and weights take positive and negative values. Then, a positive weight to an output neuron z_j from a hidden neuron y_i means an excitatory role of y_i to z_j and a negative weight means an inhibitory role. A zero weight means an irrelevant role. The gradient vector computed in Eq. (16) means such excitatory-inhibitory input patterns from pre-synaptic neurons are added through iterative error-backprop procedures. Because of the complete connections and an identical neuronal type, where a hidden neuron is located in the Fig. 2 does not matter, but each input image must have a sufficient number of hidden neurons in every hidden area to excite for its signals to reach the corresponding output neuron. The initial role assignment patterns in initial weights do matter for the final the fitting error rate, the validation error rate, and the test error rate, because gradient updates are local and inherited such initial roles. ■

Theorem 6 (Percentage luck of error-backprop): Suppose a CNN has $l > 1$ areas, A_0, A_1, \dots, A_l , connected by a cascade or a variation thereof. A_0 takes input frames $\{x \in X\}$ and A_l is the output area for classification. Suppose an area has a total of m hidden neurons that share a common receptive field R in A_0 . Consider a given input frame x . Let the percentage of the m hidden neurons that do not fire among all neurons in the same area with the same receptive field be denoted as $p(x)$. Then, the error-backprop depends on the average $\bar{p} = E_{x \in X}\{p(x)\}$ to be a reasonably small value, called *the percentage luck*.

Proof: To guide the proof, we should mention that DNs use top-k competition so that each receptive field in each area has only k neurons that fires, where k is small, e.g., top-1, for each receptive field R . Suppose a receptive field R represents a neuronal column that has n neurons. A neuronal area at level l is denoted as A_l . Every receptive field image $x \in X = A_0$ is concrete by which we means that its neurons are only pixels $\{x\}$ of a concrete example of a class C with $\bar{p}(x) = P(x \text{ fires}) \approx 50\%$ (e.g., 50% black and 50% white). Each neuron z in area A_l is abstract by which we means that it fires means an abstract class C that x belongs to, with $\bar{p}(z)$ being small corresponds to top-1 among n neurons. Then, it is necessary for the CNN to convert the most concrete representation of pixels in A_0 to more abstract representations in A_l , $l > 0$ with a low $\bar{p}(z)$. For example, in Fig. 2, we have $l = 2$ and there is no completion in the hidden area A_1 . Then error-backprop depends on that each neuron in A_2 has only a relatively smaller percentage among $n = 6$ neurons in A_1 that are positive, i.e., as the features of its particular class. The requirement of being a small percentage is due to the need for other non-firing neurons to deal with many other patterns in the same receptive field. ■

As we can expect, such a low percentage condition is rarely satisfied by a random weight vector. The more random weight vectors one uses, the better chance to hit the luck.

From Theorems 4 through 6 and their proofs, we can see that the luck of role assignment is a critical flaw of error-backprop, and so are the system parameters and the simple-minded regularization of the learning rate. Because of these key reasons, PSUTS plays a critical role to select the luckiest network from many unlucky ones after error-backprop. The more networks have trained by error-backprop, the more likely the luckiest one has a good role-assignment to start with.

There has been no lack of papers that claim to justify error-backprop does not over fit, e.g., variance based stochastic gradient decent [80], saddle-free deep network [81], drop out [82], implicit regularization during gradient flow [83]. They all address only local issues of neural networks trained by error-backprop and did not mention Post-Selections. The theory here addresses, the global role-assignment problem of random weights that no local mechanisms can deal with. This seems to be why PSUTS is necessary by error-backprop, but PSUTS is controversially fraudulent in terms of protocol—test sets are meant to test a reported system, not supposed to be used to decide which network to report from many.

IV. HOW A DN AVOIDS POST-SELECTIONS

Apparently, a brain does not use Post-Selection at all, whether UPSVS or PUUTS, because every human child must develop in a human environment to make his living. He should not be covered up and not reported, regardless how well or bad he performs. Cresceptron in the 1990s [42], [43], [16], [44], [45] and later DN [46], [84], [57], [58] were inspired by the interactive mode that brains learn though lifetime. In other words, Cresceptron and DN do not need Post-Selections. Furthermore, every DN must be ML-optimal given the same Three Learning Conditions.

A. New AI Metrics: Developmental Errors

In contrast to Post-Selections likely used by [21], [14], [19], [79], [23], [66], [24], [25], [27], [76], [29], [31] including probably AlphaGo [26], AlphaGo Zero [28], AlphaZero [85], AlphaFold [30] and MuZero [86] and many others, we define and reported developmental errors that includes all errors occurred through lifetime of each learning network:

Definition 7 (Developmental error): A Developmental Network is denoted as $N = (X, Y, W_y, Z, W_z, A)$ with sensory area X , skull closed hidden area Y and its weight space W_y , and motor area Z and its weight space W_z , and the space of architecture parameters A , where X , Y , and Z also denote the spaces of responses of X , Y and Z areas, respectively. The space of architecture parameters A includes all remaining parameters and memory of the network, other than neuronal weights, such as ages of neurons (for learning rates), neuronal patterning parameters (location and receptive fields adapted by synaptic maintenance), neuronal types (for initial connection absences among areas), and neuronal growth rates (for speed of mitosis). It runs through lifetime by sampling at discrete time indices as $N(t)$, $t = 0, 1, 2, \dots$. Start at inception $t = 0$ with supervised sensory input $\mathbf{x}_0 \in X(0)$, initial state $\mathbf{z}_0 \in Z(0)$, randomly initialized weigh vector $\mathbf{y}_0 \in Y(0)$, initial architecture $\mathbf{a}_0 \in A(0)$. At each time t , $t = 1, 2, \dots$, the network $N(t)$ recursively and incrementally updates:

$$(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{a}_t) = f(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1}, \mathbf{a}_{t-1}) \quad (17)$$

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 $N(t)$ generates a motor vector \mathbf{z}_t and is observed by the teacher and its vector difference from the desired \mathbf{z}_t^* is recorded as error e_t . The lifetime average error for each motor concept or component, from time 0 up to time t is defined as

$$\bar{e}(t) \triangleq \frac{1}{t} \sum_{i=0}^t e_i, \quad (18)$$

which is computed incrementally in terms of average developmental error $\bar{e}(t)$:

$$\bar{e}(t) = \frac{t-1}{t} \bar{e}(t-1) + \frac{1}{t} e_t. \quad (19)$$

Namely, all errors across a lifetime, at every time instance, are caught by the developmental error. In order to reach a small error, a low final error rate that a batch learning method tries to reach is not sufficient. Instead, the network must learn as fast as possible and avoid errors as much as possible at every time instance t . This is indeed important since earlier performance will shape later learning.

An optimal network that gives the lowest possible developmental error, among all possible networks under the same Three Learning Conditions, must be optimal at every time instance t throughout its life. DN is one such network. Post-Selections are useless among neural networks that give the smallest developmental error under the same Three Learning Conditions, because the maximum-likelihood optimality should give equivalent networks of the same developmental error.

However, in practice, the learning experience in the Three Learning Conditions is unlikely the same among different networks, because each physical robot that runs a network at least occupies distinct physical locations in the real world. For example, if two physical robot in the same family fight for a toy, the winner gains a winner experience and the loser may acquire a loser mentality. In other words, even if the parents of two boys are not biased toward any boys, the competition among the boys results in different learning experiences.

The developmental error is important. If a competition is based on developmental errors (such as during AIML Contests [62]), the winner is unlikely be one that uses a brute force method but has an excessive amount of computational resources and manpower. ImageNet competitions [61] are flowed also in this sense.

Although not formally defined as developmental errors, Cresceptron [16] and Developmental Networks [41], [87], [88] reported developmental errors.

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

For more a specific time period, such as the period from age t_1 to age t_2 , the average error is denoted as $\bar{e}[t_1 : t_2]$. Therefore, $\bar{e}(t)$ is a short notation for $\bar{e}[0 : t]$.

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

$$\mathbf{c}_t \triangleq (\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \quad (20)$$

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 free-state neuron that happens to be the best match is spawned that perfectly memorizes this new context regardless its randomly initialized weights. When all the free 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 [41] and for DN-2 by [87], as we will discuss below.

Note that a developmental system has two input areas from the environment, sensory X and motor Z . That is, motor Z is supervisable by the world (including teachers) but not often. 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 make an error, the motor-error that the teacher made is also recorded as the developmental error of the motor of the learner but due to the teacher.

B. Neuronal Competitions

As discussed above, error-backprop learning is without neuronal competitions. The main purpose of competition is to automatically assign roles to hidden neurons. Below, we consider two kinds of Convolution Neural Networks (CNNs), sensory networks and sensorimotor networks. A sensory network is feedforward, from sensor to motor, in computation flow and therefore is simpler and easier to understand. A sensorimotor network takes both sensor and motor as inputs and is highly recurrent and therefore more powerful.

1) *Sensory networks:* Let us first consider the case of feed-forward networks as illustrated in Fig. 3. Fig. 3(a) shows a situation where the number of samples in X is larger than the number of hidden neurons, which is typical. 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 $(\mathbf{x}, l) \in X \times L$ arrives incrementally through time, $t = 0, 1, 2, \dots$. This is the case with Cresceptron (and some other networks) which conducts incremental learning by dealing with image-label pairs one at a time and update the network 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 (also called drop-out).

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. 3(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.

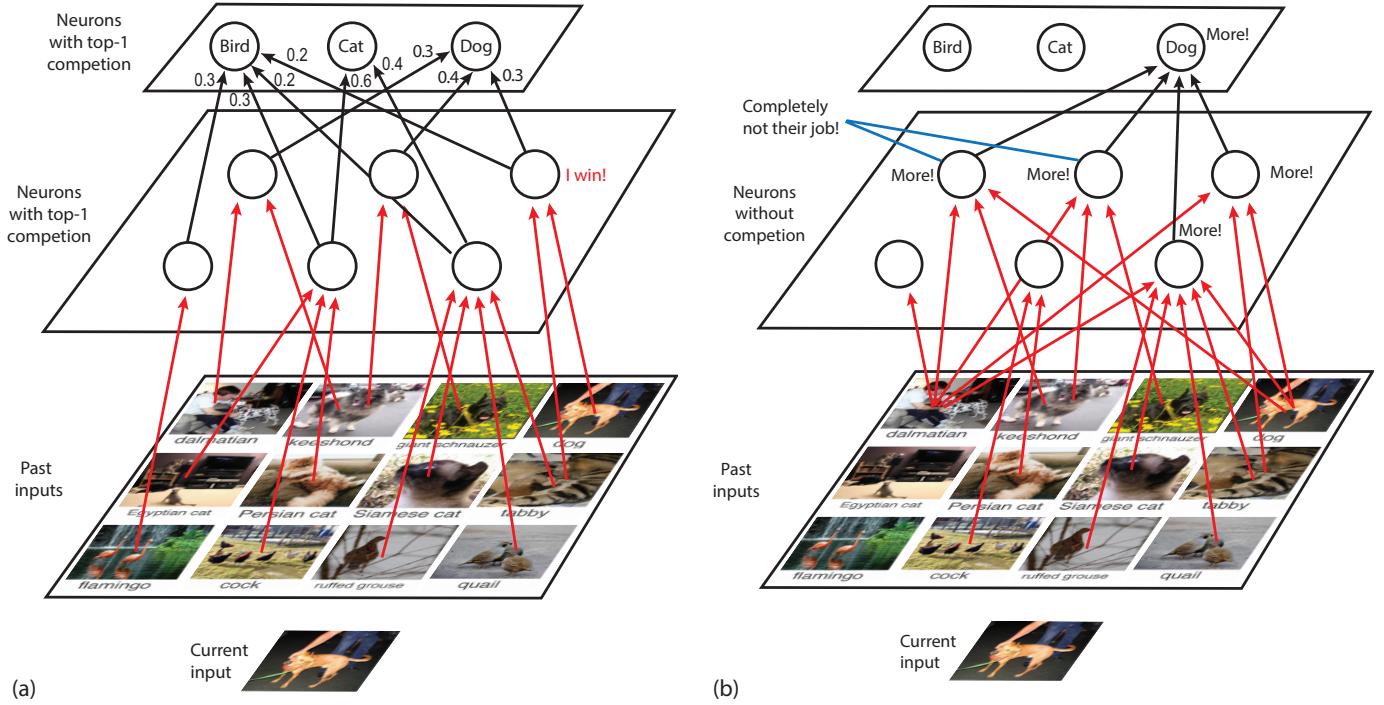


Fig. 3. How competition automatically assigns roles among hidden neurons without a central controller: The case for automatically construct a mapping $f : X \mapsto L$. (a) The number of samples in X is larger than the number of hidden neurons such that each hidden neuron must win-and-fire for multiple 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 [61].

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. 3(a), $k = 1$ for top- k competition in Y .

Likewise, suppose top-1 competition in the next higher layer, Y , namely each time only one Y neuron fires at 1 and all other Y neurons do not fire, resulting the connection patterns from the second layer Y to the next higher layer Z . In the output layer Z , top-1 competition takes place but a human teacher can supervise the pattern.

The Candid Covariance-free Incremental (CCI) Lobe Component Analysis (LCA) in Weng 2009 [67] 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 LCA vector.

Intuitively, regardless what random weights each hidden neuron starts with, as soon as it wins to fire for the first time, 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 $w_2 = 1/a = 1$ so that the new weight vector becomes the first input $r\mathbf{x}$ with $r = 1$ for the firing winner.

$$\mathbf{v} \leftarrow (1 - \frac{1}{a})\mathbf{v} + \frac{1}{a}r\mathbf{x}. \quad (21)$$

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 4.

Fig. 3(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 role problem of error-backprop in Theorem 5.

However, optimal network for incrementally constructing a mapping $f : X \mapsto L$ is too restricted, since $f : X \mapsto L$ 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 [5] 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. 3, but also a sensorimotor feature. By sensorimotor feature, we mean that the firing of each hidden neuron in Fig. 3 is determined not just by the current image σ represented by a sensory vector $x \in X$, but also the state q represented by a motor vector $z \in Z$. It is well known that a biological brain contains not only bottom-up inputs from $x \in X$ but also top-down inputs from $z \in Z$. In summary, each hidden neuron represents a sensorimotor feature in a complex brain-like network.

C. FA as sensorimotor mapping

This sensorimotor feature is easier to understand if we use the conventional symbols for (symbolic) automata. 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 [41] extended the definition the FA so that it outputs its state so the resulting FA becomes an Agent FA (AFA). Further, Weng 2015 [41] extended the action q to the machinery of Turing machine (see Fig. 4) 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 [41] proved that the control of any Turing machine is an AFA, a surprising result.

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.

We should not use symbols like σ and q , but instead sensory vectors $x \in X$ and motor vectors $z \in Z$ that are emergent as discussed above. At discrete time $t = 0, 1, 2, \dots$, we use the hidden neurons in the Y area to incrementally learn the transitions:

$$\begin{bmatrix} Z(0) \\ Y(0) \\ X(0) \end{bmatrix} \rightarrow \begin{bmatrix} Z(1) \\ Y(1) \\ X(1) \end{bmatrix} \rightarrow \begin{bmatrix} Z(2) \\ Y(2) \\ X(2) \end{bmatrix} \rightarrow \dots \quad (22)$$

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. With LCA update, [41] proved that such a DN is ML-optimal and has a constant complexity for each update $O(1)$ with a large constant, suited for real-time computation with a large memory and many neurons.

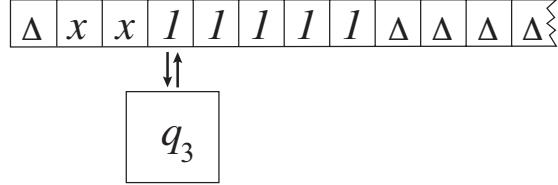


Fig. 4. A Turing machine has a tape, a read-write head, and a transition function with a current state.

D. DN as a ML-Optimal of Emergent Universal Super-Turing Machine

The traditional Turing Machine (TM) is a human handcrafted machine, as illustrated in Fig. 4.

A Universal TM (UTM) is 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 of the current general-purpose computers 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 [41] 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. The programs on the tape are learned by the Super UTM incrementally from the real world across its lifetime!

E. DN as a ML-Optimal Learning Engine for APFGP and Conscious Learning

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 those in Definition 2. This means that a DN ML-optimally learns to Autonomous Programming for General Purposes (APFGP) [57], [89]. Based on the capability of APFGP, Weng 2020 argued that APFGP is a characterization of conscious machines [58] that boots its skill of consciousness through conscious learning—being (partially) 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 degree of consciousness from infancy.

In the following, we list the DN algorithm so that we can understand APFGP is not a vague idea and how APFGP by DN avoids Post-Selections.

F. DN-2 Algorithm

Let us go through the DN-2 algorithm here so that we can see that DN is fully detail in computer implementation.

DN-2 is the latest general-purpose learning engine in the DN family. In DN-1, the allocation of neurons in each subarea of the hidden Y area is handcrafted by the designer. In DN-2, several biology-inspired mechanisms are added to automatically allocate neuronal resources and generate a dynamic and fluid hierarchy of internal representations during learning, relieving the human designer from handcrafting a concept hierarchy, beyond the rigid hierarchy in deep learning [42], [43], [16], [44], [45], [19], [79], [23], [66], [24], [25], [27], [76], [29], [31]. Namely, a DN-2 starts with simple internal representations which gradually grow to be rich and deep supported by early representations as a “brain stem”, but it is still ML-optimal conditioned on those in Definition 2.

Areas from low to high: X : sensory; Y hidden (internal); Z : motor. From low to high: bottom-up. From high to low: top-down. From one area to the same area: lateral. X does not link with Z directly.

Input areas: X and Z ; Output areas: Z ; Hidden area: Y , fully closed from $t = 0$.

- 1) At time $t = 0$, inception. Initialize the X , Y and Z areas. $\mathbf{x} \in X$ takes the first image. Set every Y neuron with random weights, zero firing age, and zero response $\mathbf{y}(0)$. Set the total number of Y neurons to be n_Y . A boundary c_Y indicates the number of active neurons ($c_Y \leq n_Y$). Set the Z area and its memory part M_Z similarly, but all concept zones take none vectors if the learner has no prenatally learned inborn “reflexes”.
- 2) For time $t = 1, 2, \dots$, repeat the following steps forever (executing steps 2a, 2b in parallel, before step 2c):
 - a) All Y neurons compute in parallel:

$$(\mathbf{y}', M'_Y) = f_Y(\mathbf{c}_Y, M_Y) \quad (23)$$

where context $\mathbf{c}_Y = (\mathbf{x}, \mathbf{y}, \mathbf{z})$, M_A denotes the memory of area A including weights and neuronal firing ages, and f_Y is the Y area function using LCA [65], [67]. If the best active Y neurons do not match the input vector well, area Y transfers new neurons to active and increment the boundary c_Y .

- b) Supervise \mathbf{z}' if the teacher likes. Otherwise, Z neurons compute the response vector \mathbf{z} and update memory M'_Z in parallel:

$$(\mathbf{z}', M'_Z) = f_Z(\mathbf{c}_Y, M_Z) \quad (24)$$

where f_Z is the Z area function using LCA [65], [67] and $\mathbf{c}_Z = (\mathbf{y}, \mathbf{z})$.

- c) Replace asynchronously: $(\mathbf{y}, M_Y, \mathbf{z}, M_Z) \leftarrow (\mathbf{y}', M'_Y, \mathbf{z}', M'_Z)$. Supervise input \mathbf{x} .

The area function f_Y in Eq.(23) and area function f_Z in Eq.(24) include two parts: (1) The computation of response vectors \mathbf{y}' and \mathbf{z}' , respectively; (2) The maintenance of memory M'_Y and M'_Z for Y area and Z area, respectively.

The ML-optimality of DN-1 and DN-2 is rooted in the optimality of LCA and extends to the entire network and entire lifetime.

G. Methods for Recursive Optimization

Given the Three Learning Conditions, at each time t , $t = 1, 2, \dots$, 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, there $\theta(t) = (\mathbf{w}_y, \mathbf{w}_z, \mathbf{a})$ where some parameters of the architecture parameter vector \mathbf{a} are hand-initialized such as the receptive fields. The maximum estimator for θ corresponds to the θ that maximizes the probability density. Regardless \mathbf{z} is imposed, \mathbf{z} is part of the parameters to be computed in a closed-form as a self-generated version:

$$(\theta^*, \mathbf{y}^*, \mathbf{z}^*) = \underset{(\theta, \mathbf{y}, \mathbf{z})}{\operatorname{argmax}} f_\theta(\mathbf{x}, \mathbf{z}). \quad (25)$$

Since the above 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 in a closed-form by the incremental version of Eq. (25) where f uses context $\mathbf{c}_{t-1} = (\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1})$:

$$(\theta_t^*, \mathbf{y}_t^*, \mathbf{z}_t^*) = \underset{(\theta_t, \mathbf{y}_t, \mathbf{z}_t)}{\operatorname{argmax}} f_{\theta_t}(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}^*, \mathbf{z}_{t-1}^*). \quad (26)$$

The DN computes the above expression for each time t in a closed form without conducting any iterations [41], [87].

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 $V = \{\dot{\mathbf{v}}_i \mid i = 1, 2, \dots, k + 1\}$ 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 $m(t)$ discussed below, activate a free neuron to fire. Eq. (21) showed that the initial weights of this free neuron is multiplied by a zero and therefore do not affect its updated weights.

Weng [41] 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. Weng [87] did the same for DN-2.

H. How DN Avoids Post-Selections but is Further ML-Optimal

Since weights are initialized randomly, how does a DN result in an equivalent network regardless the random seed? There are $k + 1$ initial neurons in the Y area, and $V = \{\dot{\mathbf{v}}_i \mid i = 1, 2, \dots, k + 1\}$ is the current synaptic vectors in Y . Whenever the network takes an input \mathbf{p} , every Y neuron computes the pre-response. If the top-1 winner in Y has a pre-response lower than almost perfect match $m(t)$, activate a free neuron to fire. The almost perfect match $m(t)$ is defined as follows:

$$m(t) = (1 - \delta)(1 - e^{-t/t_1}) \quad (27)$$

where δ is the bound of machine round-off errors, and t_1 the childhood length.

Using a mathematical induction procedure, Weng [41] proved that DN-1 computes the ML-estimator of all observations from $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ using a large constant time complexity for each time t . Weng et al. 2018 [87] proved the ML-optimality for DN-2. Since the number of transition of any Turing machine is finite, when the DN learns a Turing machine, a finite number of hidden Y neurons is sufficient for the DN to incrementally memorize exactly all the transitions observed from the Turing machine. In other words, although DN learns incrementally, such a DN is error-free for learning any complex Turing machines, including any universal Turing machines.

If the DN runs in the real world, any finite size DN is not error-free soon after inception since the number of observations from the real world is virtually unbounded, although each life is time bounded. Namely, the amount of data from the real world is so large that any practically large DN will eventually run out of free neurons that have not fired yet. From that point on, the DN is no longer guaranteed to be error-free, although could be sometimes error-free, but is still ML-optimal inside the skull conditioned on those in Definition 2. In other words, in the sense of ML, the DN is free of local minima inside the skull. That is why only one DN is sufficient for each life and the DN avoids Post-Selections. However, because the three conditions in Definition 2, a DN is not be optimal unconditionally either. For example, a better designed teaching schedule or a more appropriate physical environment may enable a DN to learn and discover rules faster and better.

I. Comparison with HMM

It is important to compare the traditional Hidden Markov Model (HMM) [90] with the ML-optimal DN. (1) The former does not have any internal representations other than the symbol based probabilities; the latter self-generates internal representations to generalize based on internal-representation based probabilities (e.g., weights). (2) The former uses batch learning but the latter uses incremental learning. (3) States in the former are symbolic, static, only partially observable for HMM, and not teachable but those in the latter are emergent, observable and directly teachable if the teacher like. (4) The former requires a batch clustering method (e.g., k-mean clustering) to initialize a static set of symbolic states, but the states/actions in the latter are incrementally taught or autonomously generated and tried. (5) Clusters of states in the former are not supported by a statistical optimality and the probability is only for state estimates but those in the latter are ML-optimal throughout the lifetime of learning, not in states/actions that the learner must produce and do not have a freedom for, but for the internal representations that the learner does have a high degree of freedom for. (6) Due to the need to compute internal representations,

the amount of computations in the latter is often higher than the former but the computational complexity is linear in time with a large constant (the number of weights of all available neurons).

V. EXPERIMENTS

A. Vision, Audition and Natural Languages

The recent experimental results of DN work here include (1) vision that includes simultaneous recognition and detection and vision-guided navigation on MSU campus walkways [91], (2) audition to learn phonemes with a simulated cochlea and the corresponding behaviors [92], (3) acquisition of English and French in an interactive bilingual environment [93], and (4) exploration in a simulated maze environment with autonomous learning for vision, path cost, planning, and selection of the least-cost plan, where all such emergent actions are either *covert* (thoughts) or *overt* (acts) [94]. The same network was used to learn these four very different tasks and task environments while each task embeds the ML-optimality of the network, under the Three Learning Conditions.

B. Error-Backprop vs. ML-Optimal DN

To show the effects of the absence of ML-optimality in CNN vs. the ML-optimality of DN, Fig. 5 shows the errors of the luckiest Convolutional Neural Network (CNN) trained by a batch error-backprop method and the errors of a DN trained incrementally. As we understand, batch learning should not be compared with an incremental learning method, because it is not a comparison on an equal footing. However, Fig. 5 shows that DN does a harder (incremental) work drastically better than CNN does an easier (batch) work. The task is real-world vision-guided navigation on the campus of Michigan State University. Because the DN is optimal in maximum-likelihood, it reaches the minimum error as soon as it has gone through the data set T once (one epoch). Later epochs correspond to reviews of the same data set T . According to the maximum-likelihood principle, the optimal estimate of the neuronal weights should not change but the ages of the neurons continue to advance. In contrast, the luckiest error-backprop trained CNN chosen from several random seeds need many epochs to reduce its errors and only very slowly. At the end of 500th epoch, the error of the luckiest CNN trained by error-backprop is still considerably higher than the full DN. Furthermore, as shown in Fig. 5, teaching invariant concepts, i.e., abstraction in Theorem 2, are used for reducing the optimal errors. For more detail, the reader is referred to [95].

C. AIML Contests

In the AIML Contest 2016, all teams are required to use a single learning engine to learn three sensory modalities, vision, audition, and bilingual natural languages acquisition, while the engine learns in “lifetime”. Although all the teams are free to choose any existing learning engines such as DN, tensorflow or other engines, all the teams chose DN engine (open source). The supplied simulated sequential data (yes, subject to the “big data” flaw) are as follows. When we let x be the image at each time instance and z be the pattern of landmark location-and-type and action of navigation, the DN became a vision-guided navigation machine. When we let x be the frame of firing pattern of hair cells in cochlea at each time instance and z be the dense states getContexts and the sparse type of sounds, the DN became a auditory-recognizer machine. When we let x be a time frame of vector of word (either English or French) and z be the language kind (neutral, English, and French) and meaning of each sentence context, the DN became a bilingual language learner and recognizer. Thus, the AIML Contest appeared to be the first contest that independently demonstrated task-nonspecificity and modality-nonspecificity by independent laboratories with the contest teams. All the teams are evaluated under the same Three Learning Conditions, e.g., the number of neurons in the engine must not exceed the same given bound. The Contest used the developmental error like the one defined here averaged over all the three contest tasks and across the three lifetimes. The *developmental error* ranked all the submitted contest entries and required all networks not to exceed the specified maximum number of neurons for each task so that the competition does not unfairly

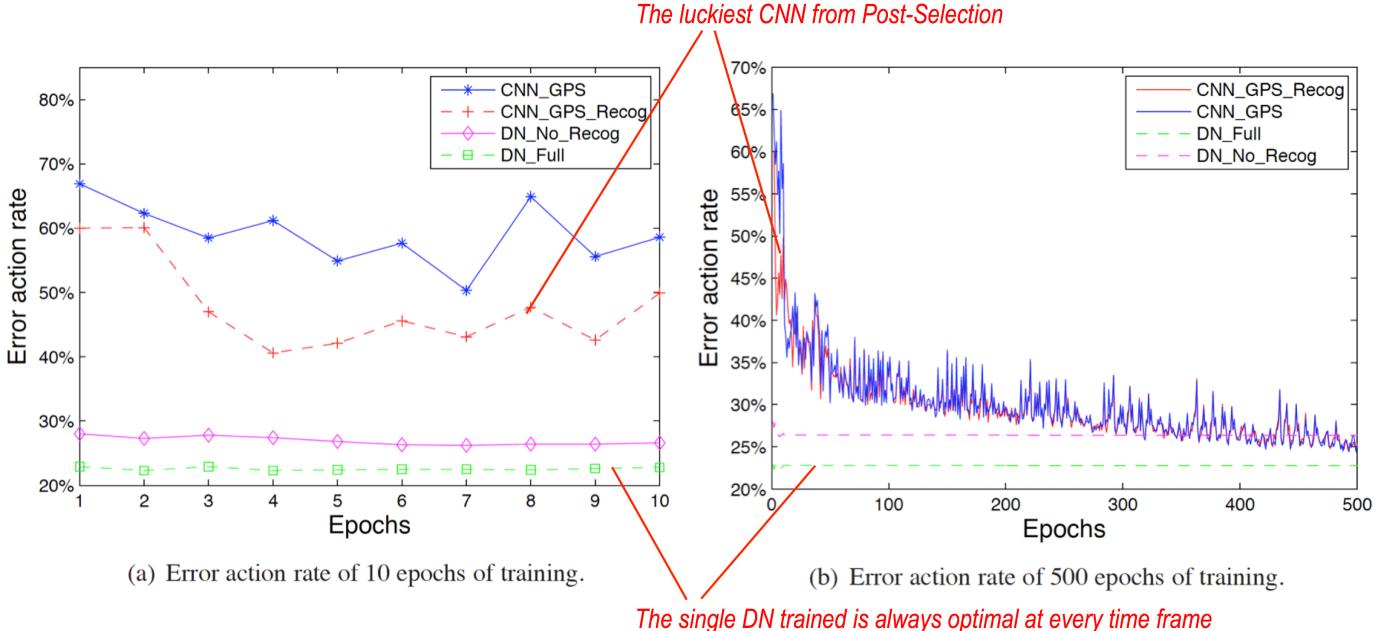


Fig. 5. Comparison of error between the luckiest CNN trained by batch error-backprop and a DN across different epochs through the training data. “Recog” and “Full” means teach where-what rules. Otherwise, data-fitting only. Adapted from [95].

favor those teams that have more computational resources at their disposal but not necessarily that their methods are more superior. All the teams have a high degree of freedom to modify the learning engine and to modify the supplied motor actions on the given data set, such as generating attentive actions on the given training set to train invariant concepts (e.g., where and what concepts) which modifies the default training experience supplied by the AIML Contest organizers but was still based on the same supplied data set. The prudent design of the AIML contests was meant to avoid the corresponding problems in ImageNet Contests [61] and many other contests.

D. GENISAMA Applications

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. Hopefully, as a APFGP platform, this new kind of human-wearable robots will be useful for practitioners to produce various kinds of intelligent auto-programmed software. The author predicts that such a new kind of AI systems will considerably alleviate the high brittleness of traditional AI software and traditional robot software in open and natural world.

Hopefully, future DN-driven robots will learn consciously and autonomously discover in the real world for Turing machine based general purposes, with relative infrequent interactions from humans similar to what parents do to their children and human teachers teach their students in classrooms. The experiments and competitions described here are for this grand goals but have not reached this experimental goal yet.

VI. CONCLUSIONS

We used intuitive terms but formal ways to discuss Post-Selections. Public and media have gained an impression that deep learning has approached or even “sometimes exceeded” human level performance on certain tasks. For example, the image classification errors from a static image set were compared with those of humans [61, A2, p242]) and the work is laudable. However, this paper raises Post-Selections, which seem to question such claims since a real human does not have the luxury of Post-Selections. The author hopes that the exposure of Post-Selections 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 for invariant concepts under the Three Learning Conditions. Some researchers have raised that it seems that those who won a competition were those who have more computational resources and manpower at their disposal. The new developmental error metrics under the Three Learning Conditions hopefully encourages future AI competitions to compare methods under the same Three Learning Conditions. Considering DN as a much-simplified model for a biological machine, it seems not baseless to guess that each biological brain is probably ML-optimal (of course in a much richer sense) across lifetime, e.g., due to the pressure to compete at every age. The Three Learning Conditions explicitly include other factors that greatly affect machine learning performances such as learning framework (e.g., task-nonspecificity, incremental learning, the robot bodies), learning experiences and computational resources. The analysis that any “big data” sets are nonscalable does not mean that we should not create, use and share data sets. Instead, we need to pay attention to the fundamental limitations of any static data sets, regardless how large their apparent sizes are.

REFERENCES

- [1] Montfort, N. *Twisty Little Passages: An Approach to Interactive Fiction* (MIT Press, Cambridge, MA, 2005).
- [2] Turing, A. M. Computing machinery and intelligence. *Mind* **59**, 433–460 (1950).
- [3] Weng, J. Symbolic models and emergent models: A review. *IEEE Trans. Autonomous Mental Development* **4**, 29–53 (2012).
- [4] Russell, S. & Norvig, P. *Artificial Intelligence: A Modern Approach* (Prentice-Hall, Upper Saddle River, New Jersey, 2010), 3rd edn.
- [5] Minsky, M. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine* **12**, 34–51 (1991).
- [6] Lenat, D. B., Miller, G. & Yokoi, T. T. CYC, WordNet, and EDR: Critiques and responses. *Communications of the ACM* **38**, 45–48 (1995).
- [7] Gomes, L. Machine-learning maestro Michael Jordan on the delusions of big data and other huge engineering efforts. *IEEE Spectrum* (Online article posted Oct. 20, 2014).
- [8] Rumelhart, D. E., McClelland, J. L. & the PDP Research Group. *Parallel Distributed Processing*, vol. 1 (MIT Press, Cambridge, Massachusetts, 1986).
- [9] McClelland, J. L., Rumelhart, D. E. & The PDP Research Group (eds.) *Parallel Distributed Processing*, vol. 2 (MIT Press, Cambridge, Massachusetts, 1986).
- [10] Krishna, R. *et al.* Visual genome. *International Journal of Computer Vision* **123**, 32–73 (2017).
- [11] Funahashi, K. I. On the approximate realization of continuous mappings by neural networks. *Neural Network* **2**, 183–192 (1989).
- [12] Poggio, T. & Girosi, F. Networks for approximation and learning. *Proceedings of The IEEE* **78**, 1481–1497 (1990).
- [13] Kohonen, T. *Self-Organizing Maps* (Springer-Verlag, Berlin, 2001), 3rd edn.
- [14] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* **36**, 193–202 (1980).
- [15] Oja, M., Kaski, S. & Kohonen, T. Bibliography self-organizing maps (som) papers: 1998 - 2001 addendum. *Neural Computing Surveys* **3**, 1–156 (2003).
- [16] Weng, J., Ahuja, N. & Huang, T. S. Learning recognition and segmentation using the Cresceptron. *International Journal of Computer Vision* **25**, 109–143 (1997).
- [17] Fukushima, K., Miyake, S. & Ito, T. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Trans. Systems, Man and Cybernetics* **13**, 826–834 (1983).
- [18] Serre, T., Poggio, T., Riesenhuber, M., Wolf, L. & Bileschi, S. High-performance vision system exploiting key features of visual cortex. *US Patent US7606777B2* (2006).
- [19] Fei-Fei, L., Fergus, R. & Perona, P. One-shot learning of object categories. *IEEE Trans. Pattern Analysis and Machine Intelligence* **28**, 594–611 (2006).
- [20] Weng, J. Dialog initiation: Modeling and: Closed skull or not? *IEEE CIS Autonomous Mental Development Newsletter* **9**, 10–11 (2012).
- [21] Werbos, P. J. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting* (Wiley, Chichester, 1994).
- [22] LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of IEEE* **86**, 2278–2324 (1998).
- [23] Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* **25**, 1106–1114 (2012).
- [24] LeCun, Y., Bengio, L. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
- [25] Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
- [26] Silver, D. *et al.* Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
- [27] Graves, A. *et al.* Hybrid computing using a neural network with dynamic external memory. *Nature* **538**, 471–476 (2016).
- [28] Silver, D. *et al.* Mastering the game of go without human knowledge. *Nature* **354–359** (2017).
- [29] McKinney, S. M. *et al.* International evaluation of an AI system for breast cancer screening. *Nature* **577**, 89–94 (2020).
- [30] Senior, A. W. *et al.* Improved protein structure prediction using potentials from deep learning. *Nature* **577**, 706–710 (2020).
- [31] Bellemare, M. G. *et al.* Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature* **588**, 77–82 (2020).

- [32] Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O. & Clune, J. First return, then explore. *Nature* **590**, 580–586 (2021).
- [33] Saggio, V. *et al.* Experimental quantum speed-up in reinforcement learning agents. *Nature* **591**, 229–233 (2021).
- [34] Willett, F. R., Avansino, D. T., Hochberg, L. R., Henderson, J. M. & Shenoy, K. V. High-performance brain-to-text communication via handwriting. *Nature* **593**, 249–254 (2021).
- [35] Slonim, N. *et al.* An autonomous debating system. *Nature* **591**, 379–384 (2021).
- [36] Mirhoseini, A. *et al.* A graph placement methodology for fast chip design. *Nature* **594**, 207–212 (2021).
- [37] Lu, M. Y. *et al.* AI-based pathology predicts origins for cancers of unknown primary. *Nature* **594**, 106–110 (2021).
- [38] Warnat-Herresthal, S. *et al.* Swarm learning for decentralized and confidential clinical machine learning. *Nature* **594**, 265–270 (2021).
- [39] Weng, J. *et al.* Autonomous mental development by robots and animals. *Science* **291**, 599–600 (2001).
- [40] McClelland, J. L., Plunkett, K. & Weng, J. Guest editorial: Convergent approaches to the understanding of autonomous mental development. *IEEE Trans. on Evolutionary Computation* **11**, 133–136 (2007).
- [41] Weng, J. Brain as an emergent finite automaton: A theory and three theorems. *International Journal of Intelligent Science* **5**, 112–131 (2015).
- [42] Weng, J., Ahuja, N. & Huang, T. S. Cresceptron: a self-organizing neural network which grows adaptively. In *Proc. Int'l Joint Conference on Neural Networks*, vol. 1, 576–581 (Baltimore, Maryland, 1992).
- [43] Weng, J., Ahuja, N. & Huang, T. S. Learning recognition and segmentation of 3-D objects from 2-D images. In *Proc. IEEE 4th Int'l Conf. Computer Vision*, 121–128 (1993).
- [44] Weng, J. Life is science (35): Did Turing Awards go to plagiarism? Facebook blog (2020). www.facebook.com/juyang.weng/posts/10158305658699783.
- [45] Weng, J. Did Turing Awards go to plagiarism? YouTube Video (2020). 1:05 hours, <https://youtu.be/EAhkH79TKFU>.
- [46] Weng, J. Why have we passed “neural networks do not abstract well”? *Natural Intelligence: the INNS Magazine* **1**, 13–22 (2011).
- [47] Ji, Z., Weng, J. & Prokhorov, D. Where-what network 1: “Where” and “What” assist each other through top-down connections. In *Proc. IEEE Int'l Conference on Development and Learning*, 61–66 (Monterey, CA, 2008).
- [48] Guo, Q., Wu, X. & Weng, J. Cross-domain and within-domain synaptic maintenance for autonomous development of visual areas. In *Proc. the Fifth Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics*, +1–6 (Providence, RI, 2015).
- [49] Felleman, D. J. & Van Essen, D. C. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex* **1**, 1–47 (1991).
- [50] Super, C. M. Environmental effects on motor development: A case of Africa infant precocity. *Developmental Medicine and Child Neurology* **18**, 561–567 (1976).
- [51] Thororoughman, K. A. & Taylor, J. A. Rapid reshaping of human motor generalization. *Journal of Neuroscience* **25**, 8948–8953 (2005).
- [52] Rizzotti, G., Riggio, L., Dascola, I. & Umlita, C. Reorienting attention across the horizontal and vertical meridians: evidence in favor of a premotor theory of attention. *Neuropsychologia* **25**, 31–40 (1987).
- [53] Moore, T., Armstrong, K. M. & Fallah, M. Visuomotor origins of covert spatial attention. *Neuron* **40**, 671–683 (2003).
- [54] Iverson, J. M. Developing language in a developing body: the relationship between motor development and language development. *Journal of child language* **37**, 229–261 (2010).
- [55] Weng, J. & Luciw, M. Brain-like emergent spatial processing. *IEEE Trans. Autonomous Mental Development* **4**, 161–185 (2012).
- [56] Weng, J., Luciw, M. & Zhang, Q. Brain-like temporal processing: Emergent open states. *IEEE Trans. Autonomous Mental Development* **5**, 89 – 116 (2013).
- [57] Weng, J. Autonomous programming for general purposes: Theory. *International Journal of Humanoid Robotics* **17**, 1–36 (2020).
- [58] 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).
- [59] Weng, J. On post selections using test sets (PSUTS) in AI. In *Proc. International Joint Conference on Neural Networks*, 1–8 (Shenzhen, China, 2021).
- [60] Weng, J. A developmental method that computes optimal networks without post-selections. In *Proc. IEEE International Conference on Development and Learning*, 1–6 (Beijing, China, 2021).
- [61] Russakovsky, O. *et al.* ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* **115**, 211–252 (2015).
- [62] Weng, J. *et al.* Emergent Turing machines and operating systems for brain-like auto-programming for general purposes. In *Proc. AAAI 2018 Fall Symposium: Gathering for Artificial Intelligence and Natural Systems*, 1–7 (Arlington, Virginia, 2018).
- [63] Ballard, D. H. & Brown, C. M. *Computer Vision* (Prentice-Hall, New Jersey, 1982).
- [64] Shapiro, L. & Stockman, G. *Computer Vision* (Addison-Wesley, New York, 2001).
- [65] Weng, J. *Natural and Artificial Intelligence: Introduction to Computational Brain-Mind* (BMI Press, Okemos, Michigan, 2019), second edn.
- [66] Karpathy, A. *et al.* Large-scale video classification with convolutional neural networks. In *Proc. Computer Vision and Pattern Recognition*, +1–8 (Columbus, Ohio, 2014).
- [67] Weng, J. & Luciw, M. Dually optimal neuronal layers: Lobe component analysis. *IEEE Trans. Autonomous Mental Development* **1**, 68–85 (2009).
- [68] Weng, J. & Luciw, M. D. Brain-inspired concept networks: Learning concepts from cluttered scenes. *IEEE Intelligent Systems Magazine* **29**, 14–22 (2014).
- [69] Wood, D. J., Bruner, J. S. & Ross, G. The role of tutoring in problem-solving. *Journal of Child Psychology and Psychiatry* 89–100 (1976).
- [70] Burr, S. Active learning literature survey. *Data Mining and Knowledge Discovery* **2**, 121–167 (1998).
- [71] Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* **60**, 84–90 (2017).
- [72] Jain, A. K. & Dubes, R. C. *Algorithms for Clustering Data* (Prentice-Hall, New Jersey, 1988).

- [73] Wang, Y., Wu, X. & Weng, J. Synapse maintenance in the where-what network. In *Proc. Int'l Joint Conference on Neural Networks*, 2823–2829 (San Jose, CA, 2011).
- [74] Guo, Q., Wu, X. & Weng, J. WWN-9: Cross-domain synaptic maintenance and its application to object groups recognition. In *Proc. Int'l Joint Conference on Neural Networks*, +1–8 (Beijing, China, 2014).
- [75] Gao, Q., Ascoli, G. A. & Zhao, L. BEAN: Interpretable and efficient learning with biologically-enhanced artificial neuronal assembly regularization. *Front. Neurorobot* 1–13 (2021).
- [76] Moravcik, M. *et al.* Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* **356**, 508–513 (2017).
- [77] Weng, J. Life is science (36): Did Turing Awards go to fraud? Facebook blog (2020). www.facebook.com/juyang.weng/posts/10158319020739783.
- [78] Weng, J. Did Turing Awards go to fraud? YouTube Video (2020). 1:04 hours, <https://youtu.be/Rz6CFIKrx2k>.
- [79] Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M. & Poggio, T. Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Analysis and Machine Intelligence* **29**, 411–426 (2007).
- [80] Sermanet, P., Kavukcuoglu, K., Chintala, S. & LeCun, Y. No more pesky learning rates. In *Proc. International Conference on Machine learning*, 343–351 (Atlanta, GA, 2013).
- [81] Dauphin, Y. N. *et al.* Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, 2933–2941 (Curran Associates, Inc., Montreal, Canada, 2014).
- [82] Srivastava, N., Hinton, G. E., Krizhevsky, K., Sutskever, I. & Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**, 1929–1958 (2014).
- [83] Poggio, T. Theoretical issues in deep networks. *Proceedings of the National Academy of Sciences* **117**, 30039–30045 (2020).
- [84] 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).
- [85] Silver, D. *et al.* A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* 1140–1144 (2018).
- [86] Schrittwieser, J. *et al.* Mastering atari, go, chess and shogi by planning with a learned model. *Science* **588**, 604–609 (2020).
- [87] 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.
- [88] Knoll, J. A. *et al.* Optimal developmental learning for multisensory and multi-teaching modalities. In *Proc. IEEE International Conference on Development and Learning*, 1–6 (Beijing, China, 2021).
- [89] Weng, J. A unified hierarchy for AI and natural intelligence through auto-programming for general purposes. *Journal of Cognitive Science* **21**, 53–102 (2020).
- [90] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE* **77**, 257–286 (1989).
- [91] Zheng, Z., Wu, X. & Weng, J. Emergent neural turing machine and its visual navigation. *Neural Networks* **110**, 116–130 (2019).
- [92] Wu, X. & Weng, J. Muscle vectors as temporally “Dense Labels”. In *Proc. International Joint Conference on Neural Networks*, 1–8 (Glasgow, UK, 2020).
- [93] Castro-Garcia, J. & Weng, J. Emergent multilingual language acquisition using developmental networks. In *Proc. International Joint Conf. Neural Networks*, +1–8 (IEEE Press, Budapest, Hungary, 2019).
- [94] Wu, X. & Weng, J. On machine thinking. In *Proc. International Joint Conf. Neural Networks*, 1–8 (IEEE Press, Shenzhen, China, 2021).
- [95] Zheng, Z. & Weng, J. Mobile device based outdoor navigation with on-line learning neural network: a comparison with convolutional neural network. In *Proc. 7th Workshop on Computer Vision in Vehicle Technology (CVVT 2016) at CVPR 2016*, 11–18 (Las Vega, 2016).

Conscious Intelligence Requires Developmental Autonomous Programming For General Purposes

Juyang Weng^{*†‡}

^{*}Department of Computer Science and Engineering,

[†]Cognitive Science Program, [‡] Neuroscience Program,
Michigan State University, East Lansing, MI, 48824 USA

Abstract—Universal Turing Machines are well known in computer science but they are about manual programming for general purposes. Although human children perform *conscious learning* (learning while being conscious) from infancy, it is unknown that Universal Turing Machines can facilitate not only our understanding of Autonomous Programming For General Purposes (APFGP) by machines, but also enable early-age conscious learning. This work reports a new kind of AI—conscious learning AI from a machine’s “baby” time. Instead of arguing what static tasks a conscious machine should be able to do during its “adulthood”, this work suggests that APFGP is a computationally clearer and necessary criterion for us to judge whether a machine is capable of conscious learning so that it can autonomously acquire skills along its “career path”. The results here report new concepts and experimental studies for early vision, audition, natural language understanding, and emotion, with conscious learning capabilities that are absent from traditional AI systems.

I. INTRODUCTION

To be conscious during adulthood, must a system be conscious from its “infancy” all the way into its “adulthood”? All animals [26], [15] conduct lifelong APFGP but traditional AI systems do not. This work argues that consciousness should not be an optional crown jewel for AI [4], [11] but a necessity for credible AI. Unconscious AI has resulted in machines that are brittle because they are unaware of themselves and the physical world around them. Machines can become highly conscious by bootstrapping its degree of consciousness through lifelong conscious learning.

Consciousness must apply to different contexts that a life experiences. Furthermore, the term involves many entities, such as environment, awareness, cognition and behavior, which is learned through lifetime based on genetically pre-positioned (i.e., developed) learning capabilities [18], [17], [4]. For example, how does a cattle or a human in Fig. 1 learn consciousness from infancy so that, after it has grown up, it can navigate autonomously through the hustle and bustle of streets to reach its home?

Can an artificial machine consciously learn to do the same and more? Weng et al. 2001 [25], which started this conference series as ICDL, proposed “autonomous mental development” as learning across lifetime that must be task-nonspecific. However, the link between “autonomous development” and “consciousness” still lacks a computational minimal set.

By “minimal set”, we mean a minimal set of mechanisms from which a machine, natural or artificial, can bootstrap its



Fig. 1. A conscious cattle and conscious humans navigate on a busy street of New Delhi, India.

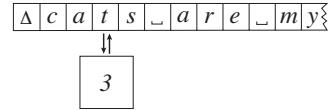


Fig. 2. An example of Turing Machine. Each cell of the tape bears only a symbol. The controller has, at each integer time, a current state (e.g., 3).

degree of consciousness from “early age” to “later age” so that a degree of consciousness is already present during learning, not only after a static batch of learning. This minimal set makes the causality of consciousness clearer.

In order to gain understanding of what that minimal set is in terms of computation, let us start with a model of computation well known in computer science but not directly related to consciousness till this work. Please skip no sections.

II. TURING MACHINES

Turing Machines, originally proposed by Alan Turing [23] in 1936, although not meant to explain consciousness, can assist us to understand how consciousness arises from computations by a machine, both natural and artificial.

A Turing Machine [7], [13], illustrated in Fig. 2, consists of an infinite tape, a read-write head, and a controller. The controller consists of a sequence of moves where each move is a 5-word sentence of the following form:

$$(q, \gamma) \rightarrow (q', \gamma', d)$$

meaning that if the current state is q and the current input that the head senses on the tape is γ , then the machine enters to next state q' , writes γ' onto the tape, and its head moves in direction d (left, right, or stay) but no more than one cell away.

Intuitively speaking, let us consider each symbol in the above 5-word expression as a “word”. Then all such 5-word expressions are “sentences”. Thus, a human-handcrafted “program” is a sequence of such 5-word sentences the Turing Machine must follow in computation. Although such sentences are not a natural language, they are more precise than a natural language.

III. UNIVERSAL TURING MACHINES

How did Turing make the above machine general-purpose? All we need is to augment the meaning of the input on the tape: The tape contains two parts, program P and data x .

In his 1936 paper [23], Turing explained in detail how a Turing Machine can be constructed to emulate program P applied to data x . This new kind of Turing Machines is now called Universal Turing Machines [7], [13]. We called it *universal* because the program P on the tape is open-ended, supplied by any users for any purposes. However, Universal Turing Machines are still not conscious.

To see the link between Universal Turing Machines and consciousness, we must break a series of restrictions in Turing Machines, as explained in the next section.

IV. EIGHT CONDITIONS FOR CONSCIOUSNESS

The eight conditions below were not well known to be necessary for consciousness. However, the APFGP capability in the title requires all of them. That said, they are still insufficient for giving rise to APFGP without the full Developmental Networks (DN) to be discussed in the next section.

To facilitate memorization, let us summarize the eight conditions in eight words: Grounded, Emergent, Natural, Incremental, Skulled, Attentive, Motivated, Abstractive, giving acronym GENISAMA. Let us explain each of them below.

Grounded: Grounded means sensors and effectors of a learner must be directly grounded in the physical world in which the learner lives or operates. IBM Deep Blue, IBM Watson, and AlphaGo are not grounded. Instead, it is humans who synthesize symbols from the physical world, and thus shield machines off from the rich physical game environments, including their human opponents.

Emergent: The signals in the sensors, effectors and all representations inside the “skull” of the learner must emerge automatically through interactions between the learner and the physical world by way of sensors, effectors, and a genome (aka developmental program). A genome is meant to fit the physical world through the entire life, not only for a specific task during the life. For example, fruit flies must do foraging, fighting and mating. Thus, task-specific handcrafting of representation in sensors, effectors, and inside the “skull” is inconsistent to consciousness. This emergence requirement rules out task-specific and handcrafted representations, such as weights duplication

in convolution used by deep learning. Likewise, an artificial genetic algorithm without lifetime learning/development does not have anything to emerge since each individual does not learn/develop in lifetime.

Natural: The learner must use natural sensory and natural motor signals, instead of human hand-synthesized features from sensors or hand-synthesized class labels for effectors, because such symbols and labels are not natural without a human in the loop. For robots, natural signals are those directly available from a sensor (e.g., RGB pixel values from a camera) and raw signals for an effector/actuator. IBM Deep Blue, IBM Watson and AlphaGo all used handcrafted symbols for the board configurations and symbolic labels for game actions. Such symbols are not natural, not directly from cameras and not directly for robot arms.

Incremental: Because the current action from the learner will affect the next input to the learner (e.g., the current action “turn left” allows it to see left view), learning must take place incrementally in time. IBM Deep Blue, IBM Watson and AlphaGo have used a batch learning method: all game configurations are available as a batch for the learner to learn. The learner is not conscious of how it has improved from early mistakes—not self-conscious.

Skulled: The skull closes the brain of the learner so that any teacher’s direct manipulations with the internal brain representations (e.g., twisting internal parameters) are not permitted. For example, how can the brain be aware of what a neurosurgeon did inside the skull?

Attentive: The learner must learn how to attend to various entities in its environment — the body and extra-body environment. The entities include location (where to attend), type (what to attend), scale to attend (e.g., body, face, or nose), as well as abstract concepts that the learner learned in life (e.g., am I doing the right thing?). IBM Deep Blue, IBM Watson and AlphaGo did not seem to think “what am I doing?”.

Motivated: The beautiful logic that a Universal Turing Machine possesses to emulate any valid program does not give rise to consciousness as we know it. By motivation, we mean that the learner must learn motivation based on its intrinsic motives, such as pain avoidance, pleasure seeking, uncertainty awareness, and sensitivity to novelty. A system that is designed to do facial recognition does not have a motive to do things other than facial recognition. IBM Deep Blue, IBM Watson and AlphaGo did not feel happy when they won a game.

Abstractive: Although a shallow definition of consciousness means awareness, full awareness requires a general capability to abstract higher concepts from concrete examples. By higher concepts here we mean those concepts that a normal individual of a species is expected to be able to abstract. Consider movie “Rain Man”: If a kiss by a lady on the lip is sensed only as “wet”, there is a lack of abstraction. A baby cannot abstract love from the first kiss, but a normal human adult is able to. Thus, abstraction requires learning.

With the above eight requirements, we are ready to discuss GENISAMA Universal Turing Machines as super machines capable of conscious learning.

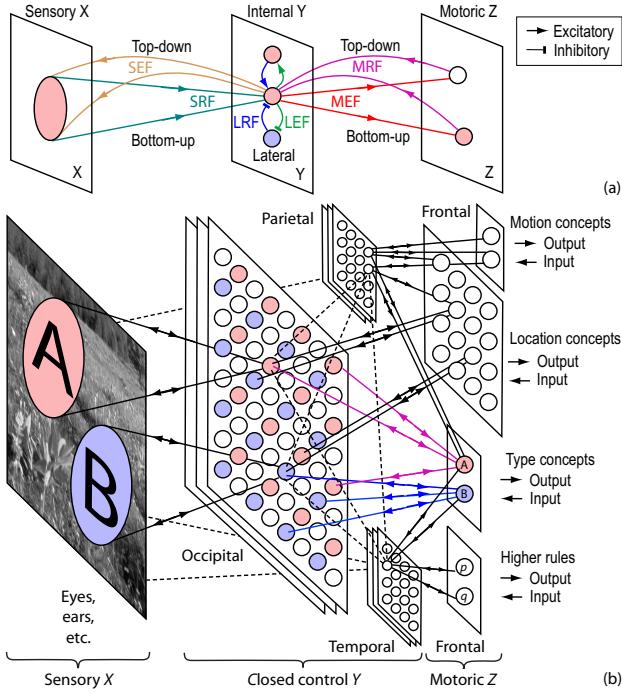


Fig. 3. A DN in (b) grows a brain Y as a two-way bridge between the sensory bank X and the motor bank Z . All the connections are learned, updated and trimmed automatically by DN. (a) A neuron's connections are highly recurrent which requires our time-unfolded explanation.

V. GENISAMA SUPER UNIVERSAL TURING MACHINES

Handcrafting a University Turing Machine is not hard. What is really hard is how to enable such a machine to grow automatically from the natural physical world so as to learn any programs and any data directly from its physical environment! We will see below how.

A DN in Fig. 3 is capable of learning any GENISAMA Universal Turing Machine. It grows one neuron at a time, to learn moves incrementally, one at a time. Such a machine is capable of APFGP, which motivates us as an alternative characterization of consciousness.

A brain is highly recurrent, meaning a neuron sends signals to many other neurons but other neurons also send signals back, directly and indirectly. This recurrence has caused great difficulties in our understanding how the brain works. We must unfold time so that the time-unfolded brain is not recurrent along the time axis. We consider five entities W, Z, Y, X, X' at times $t, t = 0, 1, 2, \dots$, as illustrated in Table I.

The first row in Table I gives the sample times t .

The second row denote the actable world W , such as a hammer acting on a nail.

The third row is the motor Z , which has muscles to drive effectors, such as arms, legs, and mouth.

The fourth row is the skull-closed brain Y . The computation inside the brain must be fully autonomous [25] without pre-given any tasks.

The fifth row is the sensor X , such as cameras, microphones, and touch sensors (e.g., skin).

The last row is the sensible world W' , such as surfaces of objects that reflect light received by cameras.

The actable world W is typically not exactly the same as the sensible world W' , because where sensors sense from and where effectors act on can be different.

Next, we discuss the rules about how a DN, denoted as $N = (X, Y, Z)$, learns from world W and W' .

Extend the tape of the Turing Machine to record images X from sensors, instead of symbols σ . Let X be the original emergent version of input, e.g., a vector that contains values of all pixels.

Extend the output from the Turing Machine (q', γ', d) to be a muscle image from motor Z , instead of symbols. Thus, the GENISAMA Turing Machine directly acts on the physical world.

Parallel computing model: We treat X and Z as external because they can be “supervised” by the physical environment as well as “self-supervised” by the network itself. The internal area Y is closed (hidden)—cannot be directly supervised by external teachers. As in the above Table, we unfold the time t and allow the network to have three areas $X(t)$, $Y(t)$, and $Z(t)$ that learn incrementally through time $t = 0, 1, 2, \dots$:

$$\begin{bmatrix} Z(0) \\ Y(0) \\ X(0) \end{bmatrix} \rightarrow \begin{bmatrix} Z(1) \\ Y(1) \\ X(1) \end{bmatrix} \rightarrow \begin{bmatrix} Z(2) \\ Y(2) \\ X(2) \end{bmatrix} \rightarrow \dots \quad (1)$$

where \rightarrow means neurons on the left adaptively links to the neurons on the right.

Note, all neurons in every column t use only the values of the column $t - 1$ to its immediate left, but use nothing from other columns t , with integers $t \geq 1$. Otherwise, iterations are required. Namely, by unfolding time in the above expression, the highly recurrent operations in recurrent DN become nonrecurrent in time-unfolded DN. Thus, the DN runs in real time and do not have to slow down waiting for any iterations.

Using Eq. (1), we outline each of the motor area Z , brain area Y , and sensory area X , for $t = 1, 2, \dots$, from an embryo, all the way to an adult, till possible death.

1. The motor area Z starting from $Z(0)$, represents many muscles signals in a developing body. Muscle cells in $Z(t)$ at time t take inputs from the $Y(t - 1)$ area and the $Z(t - 1)$, acting on the environment at time t using $Z(t)$ through self-supervision—trials, errors, and practices.

2. Concurrently, the brain Y , starting from $Y(0)$, also dynamically develops and grows. Each neuron in $Y(t)$ gets multiple inputs from all three areas, $X(t - 1)$, $Y(t - 1)$ and $Z(t - 1)$. Competition among neurons allows only few Y neurons to win. These winner Y neurons (like an expert team) at the time t column represent the firing of the brain at time t .

3. Likewise, the sensory area X , starting from $X(0)$, also develops within a developing body. What is different between the motor area Z and the sensory area X is that the former develops neurons that drive muscles (and also “feel” the world) but the latter develops receptors that sense the world.

TABLE I
UNFOLDING TIME FOR APFGP IN A DEVELOPMENTAL NETWORK

Time	0	1	2	3	4	5	6	7	...	t
Actable world W	$W(0)$	$W(1)$	$W(2)$	$W(3)$	$W(4)$	$W(5)$	$W(6)$	$W(7)$...	$W(t)$
Motor Z	$Z(0)$	$Z(1)$	$Z(2)$	$Z(3)$	$Z(4)$	$Z(5)$	$Z(6)$	$Z(7)$...	$Z(t)$
Skulled brain Y	$Y(0)$	$Y(1)$	$Y(2)$	$Y(3)$	$Y(4)$	$Y(5)$	$Y(6)$	$Y(7)$...	$Y(t)$
Sensor X	$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$	$X(5)$	$X(6)$	$X(7)$...	$X(t)$
Sensible world W'	$W'(0)$	$W'(1)$	$W'(2)$	$W'(3)$	$W'(4)$	$W'(5)$	$W'(6)$	$W'(7)$...	$W'(t)$

Now, we have the minimal set of mechanisms—Eq. (1) along with above paragraphs 1, 2 and 3—as the *Computational Model of Emergence of Consciousness* for natural and artificial machines to learn consciousness (and consciously learn) throughout lifetime:

As time goes by, the learner looks more and more aware by learning a lifetime program P and lifetime data x from its world, while an optimal GENISAMA Universal Turing Machine emerges as proven mathematically in [24]. Inside the brain this machine autonomously predicts to make a larger, more sophisticated and increasingly integrated program P for all learned purposes that apply to world data x .

In the eyes of humans, this learner becomes increasingly conscious. Although the lifelong-learned consciousness can be extremely complex, the minimal set of computational mechanisms is relatively simple for us to understand, thanks to the Universal Turing Machines.

VI. TECHNICAL DETAIL

A. Developmental Networks for Learning Consciousness

A Developmental Network is meant for consciousness because it is a holistic model for a biological brain, also fully implementable on an artificial machine.

The following section presents Developmental Network 1 (DN-1). Developmental Network 2 (DN-2) is different from DN-1 primarily in the following sense. In DN-1, each of multiple Y areas has a static set of neurons so that the competition within each area is based on a top- k principle. Namely, inhibition among neurons within each area is implicitly modeled by top- k competition.

In the DN-2, however, there is no static assignment of neurons to any regions, so that regions in the DN-2 automatically emerge, along their scales, cascade, and nesting. A major advantage of DN-2 is that a human programmer is not in the loop of deciding the distribution of X-Y-Z mechanisms, relieving human from this intractable task of handcrafting consciousness. A major disadvantage of DN-2 is that its computational explanation is too sophisticated to be included in this paper. Let us leave DN-2 out from this paper and concentrate on DN-1 below.

The hidden Y area corresponds to the entire “brain”. In the following, we assume the brain has a single area Y but it will enable many subareas to emerge.

The brain takes input from vector (z, x) , not just sensory x but also motor z , to produce an internal response vector y which represents the best match of (z, x) with one of many internally stored patterns of (z, x) :

The winner-take-all learning rule, which is highly nonlinear and simulates parallel lateral inhibition in the internal (hidden) area Y is sufficient to prove in [24] that a DN that has sufficient hidden neurons learns any Turing Machine perfectly, immediately, and error-free.

The n neurons in Y give a response vector $y = (y_1, y_2, \dots, y_n)$ of n neurons in which only the best-matched neuron fires at value 1 and all other neurons do not fire giving value 0:

$$y_j = \begin{cases} 1 & \text{if } j = \underset{1 \leq i \leq n}{\operatorname{argmax}} \{f(\mathbf{t}_i, \mathbf{z}, \mathbf{b}_i, \mathbf{x})\} \\ 0 & \text{otherwise} \end{cases} \quad j = 1, 2, \dots, n, \quad (2)$$

where f is a function that measures the similarity between the top-down weight vector \mathbf{t}_i and the top-down input vector \mathbf{z} [19] as well as the similarity between the bottom-up weight vector \mathbf{b}_i and the bottom-up input vector \mathbf{x} . The value of similarity is the inner product of their length-normalized versions [24]. Corresponding to FA, both the top-down weight and the bottom-up weight must match well for f to give a high value as inner product.

The response vector y the hidden Y area of DN is then used by Z and X areas to predict the next z and x respectively in discrete time $t = 1, 2, 3, \dots$:

$$\left[\begin{array}{c} \mathbf{z}(t-1) \\ \mathbf{x}(t-1) \end{array} \right] \rightarrow \mathbf{y}(t) \rightarrow \left[\begin{array}{c} \mathbf{z}(t+1) \\ \mathbf{x}(t+1) \end{array} \right] \quad (3)$$

where \rightarrow denotes the update on the left side using the left side as input. The first \rightarrow above is highly nonlinear because of the top-1 competition so that only one Y neuron fires (i.e., exactly one component in binary y is 1). The second \rightarrow consists of simply links from the single firing Y neurons to all firing neurons on the right side.

Like the transition function of a Turing Machine, each prediction of $\mathbf{z}(t+1)$ in Eq. (3) is called a *transition*, but now in real-valued vectors without any symbols. The same $\mathbf{y}(t)$ can also be used to predict the binary (or real-valued) $\mathbf{x}(t+1) \in X$ in Eq. (3). The quality of prediction of $(\mathbf{z}(t+1), \mathbf{x}(t+1))$ depends on how state Z abstracts the external world sensed by X . The more mature the DN is in its “lifetime” learning, the better its predictions.

The expression in Eq. (3), is extremely rich as illustrated in Fig. 3: Self-wiring within a Developmental Network (DN) as

the control of GENISAMA TM, based on statistics of activities through “lifetime”, without any central controller, Master Map, handcrafted features, and convolution.

The above vector formalization is simple but very powerful in practice. The pattern in Z can represent the binary pattern of any abstract concept — context, state, muscles, action, intent, object type, object group, object relation. However, as far as DN is concerned, they mean the same—a firing pattern of the Z area!

Namely, unified numerical processing-and-prediction in DN amounts to any abstract concepts above. In symbolic representations, it is a human to handcraft every abstract concept as a symbol; but DN does not have a human in the “skull”. it simply learns, processes, and generates vectors. In the eyes of a human outside the “skull”, the DN gets smarter and smarter.

Eq. (3)(a) shows each feature neuron has six fields in general: Sensory Receptive Field (SRF), Sensory Effective Field (SEF), Motor Receptive Field (MRF), Motoric Effective Field (MEF), and Lateral Receptive Field (LRF) and Lateral Effective Field (LEF). Eq. (3)(b) shows the resulting self-wired architecture of DN with Occipital, Temporal, Parietal, and Frontal lobes. Regulated by a general-purpose Developmental Program (DP), the DN self-wires by “living” in the physical world. The X and Z areas are supervised by body and the physical world which includes teachers.

Through the synaptic maintenance, some Y neurons gradually lose their early connections (dashed lines) with X (Z) areas and become “later” (early) Y areas. In the (later) Parietal and Temporal lobes, some neurons further gradually lost their connections with the (early) Occipital area and become rule-like neurons. These self-wired connections give rise to a complex dynamic network, with shallow and deep connections instead of a deep cascade of areas. Object location and motion are non-declarative concepts and object type and language sequence are declarative concepts. Concepts and rules are abstract with the desired specificities and invariances. DN does not have any static Brodmann areas.

B. Optimal Properties Proved for DN

If a DN can learn quickly like other normal animals, we may have to call it retarded with only a limited consciousness compared to other animals of the same age. We do not want a DN to get stuck into a local minimum, as many nonlinear artificial systems have suffered.

Fortunately, every DN is optimal in the sense of maximum likelihood, proven mathematically in [24]. Put intuitively, all DNs are optimal, given the same learning environment, the same learning experience, and the same number of neurons in the “brain”. There might be many possible network solutions some of which got stuck into local minima in their search for a good network. However, each DN is the most likely one, without getting stuck into local minima. This is because although a DN starts with random weights, all random weights result in the same network.

However, this does not mean that the learning environment is the best possible one or the number of neurons is best

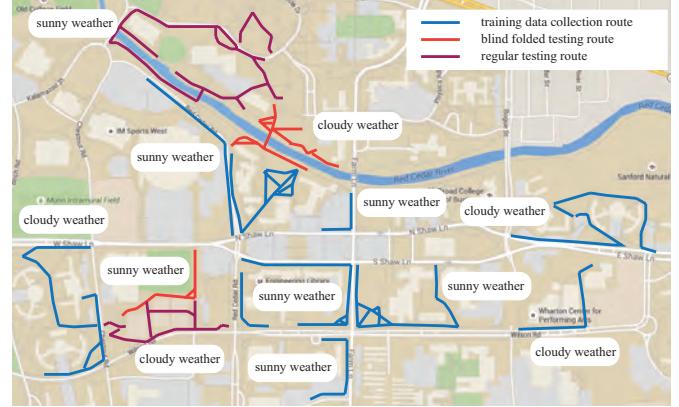


Fig. 4. Training, regular testing, and blind-folded testing sessions conducted on campus of a university under different times of day and different natural lighting conditions (e.g., there are extensive shadows in images). Disjoint testing sessions were conducted along paths that the machine has not learned. This is the first time for visual awareness to be learned by GENISAMA Turing Machines.

possible one for many lifetime tasks. Search for a better educational environment will be human challenge for their future children, both natural and artificial kinds.

C. Experiments

This seems the first time where general-purpose vision, general-purpose audition, and general-purpose natural language, as the three well-known bottleneck areas in AI has been learned by a single type of network integrated with motivational learning. Other systems include [5], [8], [22], [3], [9], [12], [16], [6].

We conducted experiments in which a learning system acts as an emergent Turing Machine that learns one of three well-recognized bottleneck problems in AI, vision, audition and natural language acquisition. Hopefully, when such systems are mature enough after “living” and “learning” in the real physical world, they look as though they have a certain degree of animal-like consciousness in the eyes of humans.

Vision from a “lifelong” retinal sequence: How does a DN become visually conscious demonstrated by its motor behaviors? Let it learn by artificially “living” in the real world!

Fig. 4 provides an overview of the extensiveness of the training, regular training, and blindfolded testing sessions. The inputs to the DN were from the same mobile phone that performs computation. They include the current image from the monocular camera, the current desirable direction from the Google Map API and the Google Directions API. If the teacher imposes the state in Z , this is treated as the supervised state. Otherwise, the DN outputs its predicted state from Z . The DN learned to attend critical visual information in the current image (e.g., scene type, road features, landmarks, and obstacles) depending on the context of desired direction and the context state. Each state from DN includes heading direction or stop, the location of the attention, and the type of object to be detected (which detects a landmark), and the

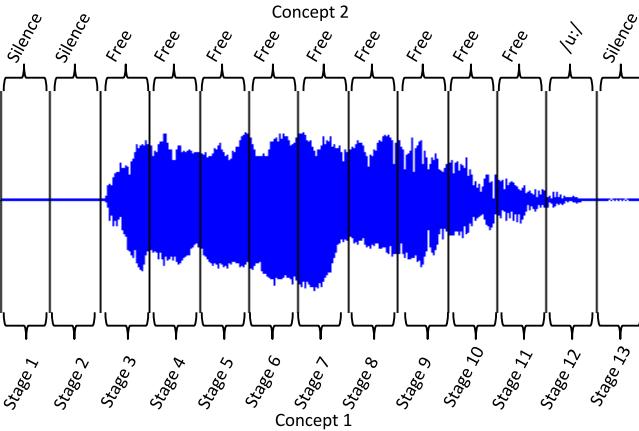


Fig. 5. The sequences of concept 1 (dense, bottom) and concept 2 (sparse, top) for phoneme /u/. The latest DNs do not need human to provide any labels. Instead, they self-supervise themselves.

scale of attention (global or local), all represented as binary patterns. None is a symbol.

Below, we will see that an auditory consciousness also uses the same DN, but using different “innate” parameters.

Audition from a “lifelong” cochlear sequence: How does a DN become auditory conscious demonstrated by its motor behaviors? Let it learn by artificially “living” in the real world!

For the audition modality, each input image to X is the pattern that simulates the output from an array of hair cells in the cochlea. We model the cochlea in the following way. The cells in the base of the cochlea correspond to filters with a high pass band. The cells in the top correspond to filters with a low pass band. At the same height, cells have different phase shifts. Potentially, such a cochlear model could deal with music and other natural sound, more general than the popular Mel Frequency Cepstral Coefficients (MFCCs) that are mainly for human speech processing. The performance will be reported elsewhere due to the limited space.

Take the phoneme /u/ as an example shown in Fig. 5. The state of concept 2 keeps as silence when inputs are silence frames. It becomes a “free” state when phoneme frames are coming in, and changes to /u/ state when first silence frame shows up at the end. At the same time, the states of concept 1 count temporally dense stages.

One may ask, what about higher consciousness such as natural language understanding?

Natural languages from a “lifelong” word sequence: How does a DN become language conscious demonstrated by its motor behaviors? Let it learn by artificially “living” in the real world! Here, we assume grounded words are emergent patterns, not symbols.

As far as we know, this seems to be the first work that deals with language acquisition in a bilingual environment, largely because the DN learns directly from emergent patterns, both in word input and in action input (supervision), instead of static symbols.

The input to X is a 12-bit binary pattern, each represents

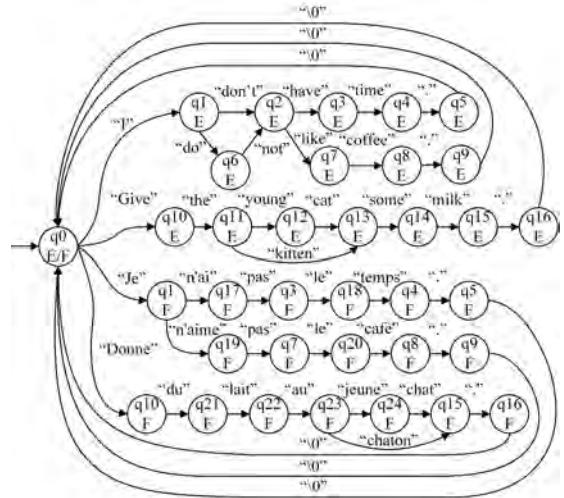


Fig. 6. The finite automaton for the English and French versions of some sentences. The DN learned a much larger finite automaton. Cross-language meanings of partial- and full-sentences are represented by the same state of meaning context q_i , $i = 0, 1, 2, \dots, 24$. See, e.g., q_1 , q_3 , q_4 , and q_5 . But the language specific context is represented by another concept: language type. The last letter is the return character that indicates the end of a sentence.

a word, which potentially can represent 2^{12} words using binary patterns. The system was taught 1,862 English and French sentences from [20], using 2,338 unique words (case sensitive). As an example of the sentences: English: “Christine used to wait for me every evening at the exit.” French: “Christine m’attendait tous les soirs à la sortie.”

The Z area was taught two concepts: language type (English, French, and language neutral, e.g., a number or name) represented by 3 neurons (top-1 firing), and the language-independent meanings as meaning states, as shown in Fig. 6. The latter is represented by 18 neurons (18-bit binary pattern), always top 5 neurons firing, capable of representing $C(18, 5) = 8,568$ possible combinations as states, but only 6,638 actual meanings were recorded. Therefore, the Z area has $3 + 18 = 21$ neurons, potentially capable of representing a huge number 2^{21} binary patterns if all possible binary patterns are allowed.

However, the DN actually observed only 8,333 Z patterns (both concepts combined) from the training experience, and 10,202 distinct (Z, X) patterns—FA transitions. Consider a traditional symbolic FA using a symbolic transition table, which has $6,638 \times 3 = 19,914$ rows and 2,338 columns. This amounts to $19,914 \times 2,338 = 46,558,932$ table entries.

But only $10,202/46,558,932 \approx 0.022\%$ of the entries were detected by the hidden neurons, representing that only 0.02% of the FA transition table was observed and accommodated by the DN. Namely, the DN has a potential to deal with n -tuples of words with a very large n but bounded by DN size, because most un-observed n -tuples are never represented. The FA transition table is extremely large, but never generated.

Without adding noise to the input X , the recognition error is zero, provided that there is a sufficient number of Y neurons. We added Gaussian noise into the bits of X . Let α represent

the relative power of the signal in the noisy signal. When α is 60%, the state recognition rate of DN is around 98%. When α is 90%, the DN has reached 0% error rate, again thanks to the power of DN internal interpolation that converts a huge discrete (symbolic) problem into a considerably smaller continuous (numeric) problem.

Emotional learning using the same network: One may wonder, does this type of consciousness enable emotion? The DN model considers emotion to belong to a wider category known in neuroscience as motivation.

Motivation is very rich [2]. It has two major aspects (a) and (b) in the current DN model. All reinforcement-learning methods other than the DN, as far as we know, are for symbolic methods (e.g., Q-learning [21], [14]) and are in aspect (a) below exclusively. The DN uses concepts (e.g., important events) instead of the rigid time-discount in Q-learning to avoid the failure of far goals.

(a) Pain avoidance and pleasure seeking to speed up learning important events. Signals from pain (aversive) sensors release a special kind of neural transmitters (e.g., serotonin [1]) that diffuse into all neurons that suppress Z firing neurons but speed up the learning rates of the firing Y neurons. Signals from sweet (appetitive) sensors release a special kind of neural transmitters (e.g., dopamine [10]) that diffuse into all neurons that excite Z firing neurons but also speed up the learning rates of the firing Y neurons. Higher pains (e.g., loss of loved ones and jealousy) and higher pleasure (e.g., praises and respects) develop at later ages from lower pains and pleasures, respectively.

(b) Synaptic maintenance—the growing and trimming of the spines of synapses—segments object/event and motivates curiosity. Each synapse incrementally estimates the average error β between the pre-synaptic signal and the synaptic conductance (weight), represented by a kind of neural transmitter (e.g., acetylcholine [27]). Each neuron estimates the average deviation $\bar{\beta}$ as the average across all its synapses. The ratio $\beta/\bar{\beta}$ is the novelty represented by a kind of neural transmitters (e.g., norepinephrine, [27]) at each synapse. The synaptogenic factor $f(\beta, \bar{\beta})$ at each synaptic spine and full synapse enables the spine to grow if the ratio is low (1.0 as default) and to shrink if the ratio is high (1.5 as default). See Fig. 3(b) for how a neuron can cut off their direct connections with Z to become early areas in the occipital lobe or their direct connections with the X areas to become latter areas inside the parietal and temporal lobes. However, we cannot guarantee that such “cut off” are 100% based on the statistics-based wiring theory here.

D. Conclusions

APFGP inside a network has a minimal set of computational mechanisms for conscious systems, natural and artificial. The new APFGP characterization is clearer than existing other characterizations for the notoriously vague term “consciousness” as we discussed in the first section. Hopefully, APFGP will give rise to richer animal-like artificial consciousness so that conscious AI receives a long-overdue

credibility for developmental AI. APFGP might also be useful as a computational model for unifying natural consciousness and artificial consciousness, due to its holistic nature backed by the new capability—APFGP of GENISAMA Universal Turing Machines. Much exciting practical work on learning consciousness remains to be done in the future.

REFERENCES

- [1] N. D. Daw, S. Kakade, and P. Dayan. Opponent interactions between serotonin and dopamine. *Neural Networks*, 15(4-6):603–616, 2002.
- [2] R. J. Dolan. Emotion, cognition, and behavior. *Science*, 298(5596):1191–1194, 2002.
- [3] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *Science*, 338:1202–1205, 2012.
- [4] J. L. Elman, E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett. *Rethinking Innateness: A connectionist perspective on development*. MIT Press, Cambridge, Massachusetts, 1997.
- [5] C. R. Gallistel. Themes of thought and thinking. *Science*, 285:842–843, 1999.
- [6] E. A. Holm. In defense of the black box. *Science*, 364(6435):26–27, April 5 2019.
- [7] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, MA, 2006.
- [8] R. Jenkins and A. M. Burton. 100% accuracy in automatic face recognition. *Science*, 319(5862):435, 2008.
- [9] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349:255–260, July 17 2015.
- [10] S. Kakade and P. Dayan. Dopamine: generalization and bonuses. *Neural Network*, 15:549–559, 2002.
- [11] C. Koch. What is consciousness? *Scientific American*, 318(6):60–64, June 2018.
- [12] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350:1332–1338, 2016.
- [13] J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw Hill, Boston, MA, 3rd edition, 2003.
- [14] V. Mnih and et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.
- [15] P. R. Montague, P. Dayan, C. Person, and T. J. Sejnowski. Bee foraging in uncertain environments using predictive Hebbian learning. *Nature*, 377:725–728, 1995.
- [16] M. Moravcik and et al. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356:508–513, 2017.
- [17] J. Piaget. *The Construction of Reality in the Child*. Basic Books, New York, 1954.
- [18] J. Piaget and M. Cook. *The Origins of Intelligence in Children*. International Universities Press, Madison, New York, 1952.
- [19] Y. B. Saalmann, I. N. Pigarev, and T. R. Vidyasagar. Neural mechanisms of visual attention: How top-down feedback highlights relevant locations. *Science*, 316:1612 – 1615, 2007.
- [20] R. Scriven, G. Amiot-Cadey, and Collins. *Collins French grammar*. HarperCollins, Glasgow, 2011.
- [21] R. S. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, Cambridge, Massachusetts, 1998.
- [22] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331:1279–1285, 2011.
- [23] A. M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math. Soc., 2nd series*, 42:230–265, 1936. A correction, *ibid*, 43, pp. 544–546.
- [24] J. Weng. Brain as an emergent finite automaton: A theory and three theorems. *International Journal of Intelligent Science*, 5(2):112–131, 2015.
- [25] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2001.
- [26] K. Wynn. Addition and subtraction by human infants. *Nature*, 358:749–750, 1992.
- [27] A. J. Yu and P. Dayan. Uncertainty, neuromodulation, and attention. *Neuron*, 46:681–692, 2005.