

# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,700

Open access books available

182,000

International authors and editors

195M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.  
For more information visit [www.intechopen.com](http://www.intechopen.com)



# Deep Learning Misconduct and How Conscious Learning Avoids it

Juyang Weng

## Abstract

“Deep learning” uses Post-Selection—selection of a model after training multiple models using data. The performance data of “Deep Learning” have been deceptively inflated due to two misconducts: 1: cheating in the absence of a test; 2: hiding bad-looking data. Through the same misconducts, a simple method Pure-Guess Nearest Neighbor (PGNN) gives no errors on any validation dataset  $V$ , as long as  $V$  is in the possession of the authors and both the amount of storage space and the time of training are finite but unbounded. The misconducts are fatal, because “Deep Learning” is not generalizable, by overfitting a sample set  $V$ . The charges here are applicable to all learning modes. This chapter proposes new AI metrics, called developmental errors for all networks trained, under four Learning Conditions: (1) a body including sensors and effectors, (2) an incremental learning architecture (due to the “big data” flaw), (3) a training experience, and (4) a limited amount of computational resources. Developmental Networks avoid Deep Learning misconduct because they train a sole system, which automatically discovers context rules on the fly by generating emergent Turing machines that are optimal in the sense of maximum likelihood across a lifetime, conditioned on the four Learning Conditions.

**Keywords:** artificial intelligence, machine learning, deep learning, ChatGPT, postselection, research misconduct, developmental networks, consciousness

## 1. Introduction

Artificial Intelligence (AI) research dates back at least to early 1910 when Leonardo Torres y Quevedo built a chess end-game player called EI 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 considering what is now called the Turing Test which 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 chapter, we will focus on generalization.

All AI methods fall into three broad schools [3], symbolic, connectionist, and developmental, although many published methods are a mixture of them.

Intuitively, Post-Selections mean first training multiple systems each starting with random weights, and then selecting a trained system based on a validation set, test set, or both. The Post-Selections are rooted in rigid representations in the symbolic school—symbols and rigid architectures. The developmental school, motivated by brain development across a lifetime, avoids Post-Selections since every life must successfully develop and must not be dropped by a Post-Selection. Therefore, let us first discuss the three AI schools.

## 1.1 Symbolic school

Symbols are used in many AI methods (e.g., states in HMMs (Hidden Markov Models), nodes in Graphical Models, and attributes in SLAM (Simultaneous Localization And Mapping)). 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 a 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 the 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 1.1 (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 chapter, the author 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, *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].  $X$  may contain 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  $L$ , to learn

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

Without the 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 [13–15].

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 that are strictly “unsupervised” used by the 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 [17–19] or implicitly as “weakly supervised” [19]. This author called them “skull-open” [20].
- B. Random initialization plus error-backprop: Locally train multiple networks. Each network starts from a different set of parameters (e.g., weights and architecture hyperparameters). Then, each network is trained using a gradient-based method, either error-backprop or value-backprop. After the training, post-select the luckiest network based on a validation set, or the union of a validation set and a test set. The former is called the Post-Selection Using Validation Set

(PSUVS). The latter is called Post-Selection Using Test Set (PSUTS). Report the luckiest network only [21–38]. Many publications do not report the post-selection stage at all, with very few exceptions [27, 33].

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 luck 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) seem 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 the 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.

Reinforcement learning does not address the Post-Selection problems. The problems of Post-Selection discussed below also defeat reinforcement learning (e.g., Q-learning) that uses symbolic states (which added values). For the problems of symbolic states in reinforcement learning, see criticisms in Ref. [42] which presents biologically inspired reinforcement learning in the Developmental Networks using emergent (vector) states.

### 1.3 Developmental school

The main thrust of the Developmental School, formally presented in 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 the 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. [16, 43, 44] 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 Refs. [45, 46] 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 an unfired fashion using a neural network started from Developmental Networks (DNs) [47] whose experimental embodiments range from Where-What Networks 1 (WWN-1) [48] to WWN-9 [49]. 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.

Note that  $Z(t - 1)$  here is extremely important since it corresponds to the state of a Turing machine. Namely, all the errors that occurred during any time of each life are 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 [50–54]. This new formulation is meant to model not only the brain’s spatial processing [55] and temporal processing [56], but also Autonomous Programming for General Purposes (APFPG) [57, 58]. Based on the APFPG capability, the AI field seems to have a powerful yet general-purpose framework toward conscious machines [59–62].

We need to consider two factors: (A) Space: Because  $X$  and  $Z$  are vector spaces of sensory images and muscle neurons, we need internal neuronal feature space  $Y$  to deal with sub-vectors in  $X$  and  $Y$  and their spatial hierarchical features. (B) Time: Furthermore, considering symbolic Markov models, we also need further to model how  $Y$ -to- $Y$  connections enable something similar to higher and dynamic order of time in Markov models. With the two considerations (A) Space and (B) Time, the above lifetime mapping in Eq. (3) is extended to:

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

If we define  $\mathbf{c}(t) \in C(t) = X(t) \times Y(t) \times Z(t)$ , Eq. (4) is extended to:

$$f : C(t - 1) \mapsto C(t), \quad (5)$$

It is worth noting that the  $Y$  space is inside a closed “skull” so it should not be directly supervised. However, many authors supervise  $Y$  in their neural networks (skull-open). This framework deals with space and time in a unified fashion using a neural network such as DN [47] whose experimental embodiments range from Where-What Network WWN-1 to WWN-9. The DN has attacked AGI (Artificial General Intelligence) problems including vision, audition, natural language acquisition, and planning as emergent Turing machines [41]. DN-2 and DN-3 are new versions of DN. It is worth noting that  $Z(t - 1)$  here is extremely important since it corresponds to the state of a Turing machine as we will see below.

Asim Roy [63] argued that some parts of the brain control other parts. Here, the area  $Z$  could be treated as a regulatory “controller” that regulates hidden neurons in the  $Y$ , e.g., as sensorimotor rules. In neuroscience, there have been many published

models that handcraft areas as top-down regulators. In the DN model below, such areas must be automatically generated and refined ML-optimally inside the closed skull across a lifetime.

In terms of performance evaluation, all the errors that occurred during any time in Eq. (4) 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 [50–54, 64]. This new symbol-free formulation is necessary to model not only the brain’s spatial processing [55] and temporal processing [56], but also Autonomous Programming for General Purposes (APFPG) [58]. Based on the APFPG capability, we open the door toward the next step—conscious learning [59–62]—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. 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.

Here, the chapter presents a controversial practice called Post-Selection. This report also explains why the inventor’s brain model (Developmental Networks, DNs) avoids Post-Selection. Using Post-Selections means at least the corresponding project wastes much resource of computation and manpower and the superficial error of the reported system is misleading because the system does not give a similar error on new data sets. This invention not only raised a controversy but also presented a solution to avoid Post-Selections.

Since 2012, AI has attracted much attention from the public and media. A large number of projects in AI have published [24–31, 65–67]. If the authors of these projects use the method of this report, they could benefit much for reducing the time and manpower used to reach their target systems as well as improving the generalization powers of their target systems.

This is the journal archival version of the earlier conference papers [68, 69] with significantly refined additional material and analysis.

In the following, we will discuss Post-Selection in Section 1.2. Section 1.3 addresses why error-backprop algorithms suffer from severe local minima problems. Section 1.4 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 1.5 discusses experiments. Section 1.6 provides concluding remarks.

## 2. Post-selections

In statistics, inference post a model selection [70] means from a given data set, a human manually selects a model, and then, the selected model is applied to the data set to conduct inference. Although this practice is allowed as a computer-assisted manual process in statistics, post-selection is inconsistent with the well-accepted principle of AI, since AI is not widely considered a manual process. The Post-Selection issue here is very different from the post-model selection topic in statistics.

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 four Learning Conditions that any fair comparisons of AI methods should take into account.

## 2.1 The four 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 competition about how many resources a group has at its disposal, regardless of how many networks have been trained and discarded, and how much time the training takes.

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

*Definition 1.2 (The Four Learning Conditions). The Four Learning Conditions for developing an AI system are: (1) A body including sensors and effectors, (2) a set of restrictions of learning framework, including whether task-specific or task-nonspecific, batch learning or incremental learning; (3) a training experience and (4) a limited amount of computational resources including the number of hidden neurons.*

The competing standard of the ImageNet competitions [71] did not include any of these four conditions. The AIML Contests [72] considered all the Four Learning Conditions in performance evaluation. In the following subsection, we discuss why task-nonspecificity and incremental mode should be considered in any comparisons.

## 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 Four Learning Conditions, the task-nonspecific learning paradigm is significantly different from the task-specific traditional AI paradigm as explained in Weng et al. [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 [73, 74] for line detection which makes every pixel that is detected as edge cast votes for lines of all possible orientations  $\theta$  and distances  $d$  from the origin that go through the pixel. Then, the top two “peaks” of line parameters ( $\theta, d$ ) 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 disappear due to weather or road conditions, leading to a brittle system. Human brains appear to be more resilient.

Even a highly specific task in ImageNet [71] cannot be done well if a learning system is task-specific—restricted to ImageNet task specifications. For example, why “stripe shirts” are classified as “zebra”? Because ImageNet data sets are meant to drop “stripe shirts” in favor of the “zebra” class. Recognition must recognize “wholes” from “parts,” but ImageNet annotations do not provide parts annotations. All such information and more are beyond the ImageNet specifications.

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 the system’s learned context  $q$  [75] 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 purpose in the sense of universal Turing machines. Any universal Turing machine is of general purpose because it can read any program written for any purpose and run it for that purpose. Any neural network that learns a universal Turing machine becomes of general purpose 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 the potential to be more robust than a world-model-based approach. The task-nonspecific approach typically uses a neural network to learn because of the need to learn vector mapping function  $f(\mathbf{z}, \mathbf{x}) = \mathbf{z}'$ . We will discuss internal response vector  $\mathbf{y}$  in Section 1.4 but task-nonspecificity holds without  $\mathbf{y}$ .

It is worth noting that providing a training set to all teams does not mean the same training experience, e.g., incremental learning vs. batch learning and the number of times to pass through the data set. See more discussion below. On the other hand, superficially stating what processors and accelerators were used does not address the network size problem as the critical computational resource here.

### 2.3 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, 24, 76] that uses a gradient-based method to find a local minimum in error.

As we will discuss in Section 1.3, 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 1.1 below.

In contrast, all developmental methods cited here use incremental learning modes 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 [77].

However, the batch and incremental learning modes are not capability-equivalent [77]. The former requires all sensory inputs to be available in 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 is technically flawed. We have the following theorem:

**Theorem 1.1 (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 the head left or right will result in a different image sensed. Therefore, all static “big data” sets violate the sensorimotor recurrence.  $\square$

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. Because vision requires a high level  $l$  to understand natural scenes with an 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 rules (e.g., finger-parts and hand-whole) instead of simple-minded pattern recognition of sensory images.

**Theorem 1.2 (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  $l^k$  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  $l^k = 2^2 = 4$  observations, corresponding to 4 rows in the truth table of logic OR). Since  $f(l, k) = l^k$  is an exponential function in  $k$ ,  $l^k$  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 Refs. [55, 78].

Section 1.4 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 [79].

Theorem 1.2 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 [76]. 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, 78]. 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 [80]) (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 Ref. [56] explains how each concept state reduces the

number of samples to be learned from an exponential  $l^k$  down to only  $kl$  (see Fig. 6 of Ref. [56] for intuition where  $k = 10$  and  $l = 3$ ). Thus, Section 1.4 not only addresses the non-scalability problems in this section, but is also necessary for conscious learning whose theory was first published in Weng [59] followed by more details [60–62], but animal-level conscious robots that are multi-sensory and multi-motor have not yet been demonstrated. The availability of real-time learning brainoid chips is a current bottleneck.

## 2.4 Fit, validation, and test errors

Given an available data set  $D$ ,  $D$  is divided by a partition  $P = F \cup V \cup T$  into three mutually disjoint sets, a fit set  $F$ , a validation set  $V$ , and a test set  $T$  so that

$$D = F \cup V \cup T. \quad (6)$$

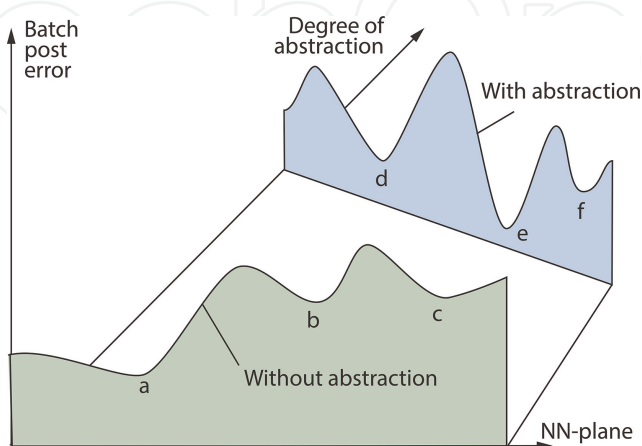
Two sets are disjoint if they do not share any elements. The validation set is possessed by the trainer, the test set should not be possessed by the trainer since the test should be conducted by an independent agency. Otherwise,  $V$  and  $T$  become equivalent.

As we will see in Section 1.3, 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 fit set, called fit error, that the error-backprop training intends to minimize locally. That is how the multiple sets of random weight hyper-parameter 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

$$\{N(\mathbf{a}_i, \mathbf{w}_j) \mid i = 1, 2, \dots, k, j = 1, 2, \dots, n\}. \quad (7)$$

Error-backprop locally and numerically minimizes the fit error  $f_{i,j}$  on the fit set  $F$ .

**Figure 1** gives a 1D illustration of the effect of abstraction. If the architecture of a neural network is inadequate e.g., pure classification through data fitting in Eq. (1),



**Figure 1.**

The effect of degree of abstraction by a neural network. The horizontal axis indicates the possible value of the parameters of a neural network, denoted as NN-plane. The 1-D here corresponds to the high-dimensional parameters of the neural network (e.g., 60-million dimension in [81]). The vertical axis is the batch post-selection error of the corresponding trained network.

the manifold of post error corresponds to that of “without abstraction” (green) in **Figure 1**. Different positions along the horizontal axis (NN-plane) correspond to different parameters of the same type of neural network. The lowest point on the green manifold is labeled “a,” but as we will see below, the smallest post error is missed by all error backprop methods since it typically does not coincide with a pit in the training data set. This is true because the test set  $T$  and the fitting set  $F$  are disjoint, but the fitting error is based on the fitting set  $F$  but the post error is based on the test set  $T$ . The height of a curve is the system fit error  $e_{ij}$  of a particular trained network  $N(\mathbf{a}_i, \mathbf{w}_j)$ . We use the term “life” to indicate the entire process of a system’s learning.

In **Figure 1**, the blue manifold is better than the green manifold, because the lowest point “e” on the blue manifold is lower than the lowest point “a” on the green manifold. They correspond to different mapping parameter vector definitions for  $\mathbf{a}$ . For example, the green manifold and the blue manifold correspond to Eqs. (1) and (3), respectively.

Note that **Figure 1** only considers batch post errors in batch learning, but Eqs. (3) and (4) deal with incremental learning.

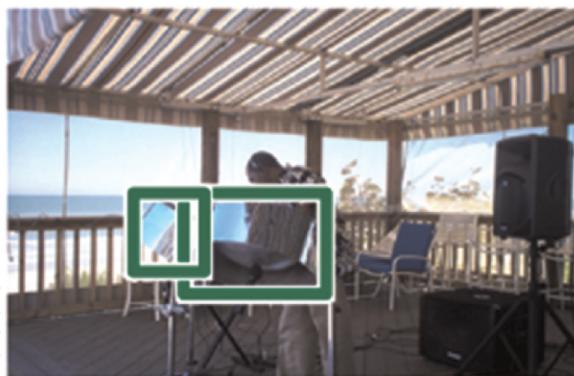
Given a defined architecture parameter vector  $\mathbf{a}$ , each searched  $\mathbf{a}_i$  as a guessed  $\mathbf{a}$  will also give a very different manifold in **Figure 1**, where for simplicity, the manifold is drawn as a line. In general, the worse a guessed  $\mathbf{a}_i$  is, the higher the corresponding position on the manifold but the amounts of increase at different points of the manifold are not necessarily the same since the manifold depends also on the test set  $T$ .

Graves et al. [27] seems to have mentioned that the number of trained systems is at least  $n = 20$ . Saggio et al. [33] reported that  $n$  is at least 10, 000. Krizhevsky & Hinton [81] 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 an example of  $l^k = k = 3^{10} = 59049$  and  $n = 20, l^k n \approx 1\text{M}$  networks must be trained, a huge number that require a lot of computational resources to do the number crunching and a lot of manpower to manually tune the range of hyper-parameters.

**Definition 1.3** (Distribution of fit, validation, and test errors). *The distributions of all  $kn$  trained networks’ fit errors  $\{f_{ij}^k\}$ , 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  $P = F \cup V \cup T$ . 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. Consider a highly specific task of recognizing patterns like those in annotated windows in **Figure 2**. This is a simplified case of the three tasks—recognition (yes or no, learned patterns at varied locations, and scales), detection (presence of, or not, learned patterns), and segmentation (of recognized patterns from input) from natural cluttered scenes dealt with by the first deep learning networks for 3D—Cresceptron [16]. Later data sets like ImageNet [71] contain many more samples.

**Definition 1.4** (Pure-Guess Nearest Neighbor, PGNN). *PGNN method stores all available data  $D$ , the fit set  $F$ , the validation set  $V$ , and the miscalled test set  $T$ . To deal with the ImageNet Competitions in **Figure 2**, the method uses the window-scan method probably first proposed by Cresceptron [16]. Given the query input from every scan window, PGNN finds its nearest-neighbor sample in  $D$  and outputs the stored label. PGNN perfectly fits  $F$ .*



**Figure 2.** Two annotated windows for an object class labeled as “steel drum” for single object localization. Figure courtesy of Ref. [71].

For samples in  $V$  and  $T$ , PGNN randomly and uniformly guesses a label from the label set  $L$  using Post-Selection and stores the guessed label.

From a fit set  $F$  and a Post set  $P = V \cup T$ , the PGNN algorithm is denoted as  $G(F, P, e)$ , where  $e$  is a seed for a pseudo-random number.

The training stage of PGNN:

The first step Fit( $F, B$ ): Store the entire fitting set  $F = \{(\mathbf{s}, l)\}$  into database  $B$ , where  $\mathbf{s}$  and  $l$  are the normalized samples and the label from the corresponding annotated window  $\mathbf{w}$ , respectively. The window scan has a pre-specified position range for row and column ( $r, c$ ) of a scan window, and a pre-specified range for the scale of the window. The window scan tries all the locations and all the scales in the pre-specified ranges. For each window  $\mathbf{w}$ , Fit crops the image at the window and normalizes the cropped image into a standard sample  $\mathbf{s}$ . All standard samples in  $B$  have the same dimension as a vector in row-major storage.

The second step Post( $P, L, e, B$ ): From every query image  $\mathbf{q} \in P$ , for every scan window  $\mathbf{w}$  for  $\mathbf{q}$ , compute its standard sample  $\mathbf{s}$ . If  $\mathbf{s}$  is new, guess a label  $l$  for  $\mathbf{s}$  to generate  $(\mathbf{s}, l)$ , where  $l$  is randomly sampled from  $L$  using a uniform distribution, identically independently distributed. Store  $(\mathbf{s}, l)$  into database  $B$ . While Post( $P, L, e, B$ ) is not good enough on  $P$ , run Post( $P, L, e, B$ ) using the returned new seed  $e$ .

Each run of Post corresponds to a new “Deep Learning” network where each network starts from a new random set of weights and a new set of hyperparameters.

The performance stage of PGNN:

Run( $P, B$ ): For every query image  $\mathbf{q} \in P$ , for every scan window  $\mathbf{w}$  for  $\mathbf{q}$ , compute its standard sample  $\mathbf{s}$ . Find its nearest sample  $\mathbf{s}^*$  from  $B$ , and output the stored label  $l$  associated with the nearest neighbor  $\mathbf{s}^*$ .

PGNN uses a lot of space and time resources for over-fitting  $F$  and  $P$ . It randomly guesses labels for  $P = V \cup T$  until all the guesses are correct. Therefore, it satisfies the required error for  $V$  and  $T$ , as long as the human annotation is consistent. PGNN here is slower than NNWT in Ref. [82] which interpolates from samples in  $F$  until the distance is beyond the threshold (a hyper-parameter). But PGNN is simpler for our explanation of misconduct since it drops the threshold in NNWT.

A typical neural network architecture has a set of hyperparameters represented by a vector  $\mathbf{a}$ , where each component corresponds to 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. 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 hyperparameters, we need to validate the sensitivity of the system error to  $x$ . With the uncertainty of  $x$ , we estimate its initial value as the mean  $\bar{x}$ , positively perturbed estimate  $\bar{x} + \sigma_x$  ( $\sigma$  is the estimated standard deviation of  $x$ ) and negatively perturbed estimate  $\bar{x} - \sigma_x$ . If each scalar hyperparameter 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 the nearest distance between a sample in  $V$  and the nearest neighbor in  $F$  and the  $\sigma_d$  be estimated by the standard deviation of these nearest distances.

Let us define the Post-Selection. Ideally, test sets should not be in the possession of authors, e.g., during a blind test. However, this is often not true since the authors may use publically available data sets that include test sets. Suppose that the validation sets and the test sets are in the possession of the human programmer.

**Definition 1.5 (Post selection).** *A human programmer trains multiple systems using the fit set  $F$ . After these systems have been trained, the experimenter 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.*

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-selections:

## 2.5 Post-selections

A post-selection can use the validation set  $V$  or the test set  $T$ . However, if both sets are in the possession of the human programmer, the difference between  $V$  and  $T$  almost totally vanishes under post-selections.

### 2.5.1 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*:

$$e_{i^* j^*} = \min_{1 \leq i \leq k} \min_{1 \leq j \leq n} e_{i,j} \quad (8)$$

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.

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

## 2.6 Cross-validation

The above PSUVS is an absence of cross-validation [83]. Originally, the cross-validation is meant to mitigate an unfair luck in a partition of the dataset  $D$  into a fit set  $F$  and a test set  $T$  (empty validation set). For example, unfair luck is such that every point in the test set  $T$  is well surrounded by points in the fit set  $F$ . But such luck is hardly true in reality.

To reduce the bias of such luck, an  $n$ -fold cross-validation protocol,  $n \geq 2$ , divides the data set  $D$  into  $n$  subsets of the same size and conducts  $n$  experiments. The term “cross” refers to switching the roles of fit and test 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 fit 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:

$$\bar{e} = \frac{1}{n} \sum_{i=1}^n e_i \quad (9)$$

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

The  $n$  different numbers here show a distribution  $\{e_j\}$  to indicate how sensitive the error is to luck, such as the number of partition pairs between a fit 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  $\{e_j\}$ .

Although post-selections have been used by many different kinds of AI methods, let us discuss neural networks as a common type of learning agent.

## 2.7 Types of lucks in a neural network

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

*Type-1 order lucks:* The luck in a partition  $P_i$  into a fit set  $F_i$  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 luck 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 been 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 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 fit set  $F$ , this chapter 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. [81] 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  $\sigma$  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 around  $1/\sqrt{2}$ , since the expected sample variance of  $n$  random numbers is  $\sigma^2/(n-1)$ .

*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 luck of  $\mathbf{a}_j$  might not capture the internal rules of the fit set  $F_j$ , this chapter 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 1.4 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 [77].

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 luck 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 the 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  $\mathbf{w}_i$  lead to the function-equivalent neural network under certain conditions. For example, in top- $k$  competition, with  $k = 1$  different  $\mathbf{w}_i$ 's give the same neural network and with  $k > 1$  different  $\mathbf{w}_i$ 's 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 Four 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 are automatically and optimally determined by each neuron, not handcrafted, as in LCA [77]. The synaptic maintenance automatically adjusts all receptive fields [84, 85] 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 luck, e.g., in ImageNet Contest [71]. Namely, errors occurred during batch training of the network before the network is finalized and how long the training takes is not reported. Below, **Figure 3** 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 that 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 hyperparameters  $\mathbf{a}_j$ . 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 luck on each vector  $\mathbf{w}_i$ , an average of  $e_{i,j}$  over  $n$  values of  $i$  in Eq. (8) should be used instead of the minimum in Eq. (15). This leads to *the random-weights validated error for the luckiest architecture* from PSUVS:

$$\mathbf{a}^* = \arg \min_{1 \leq j \leq k} \frac{1}{n} \sum_{i=1}^n e_{i,j}. \quad (10)$$

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

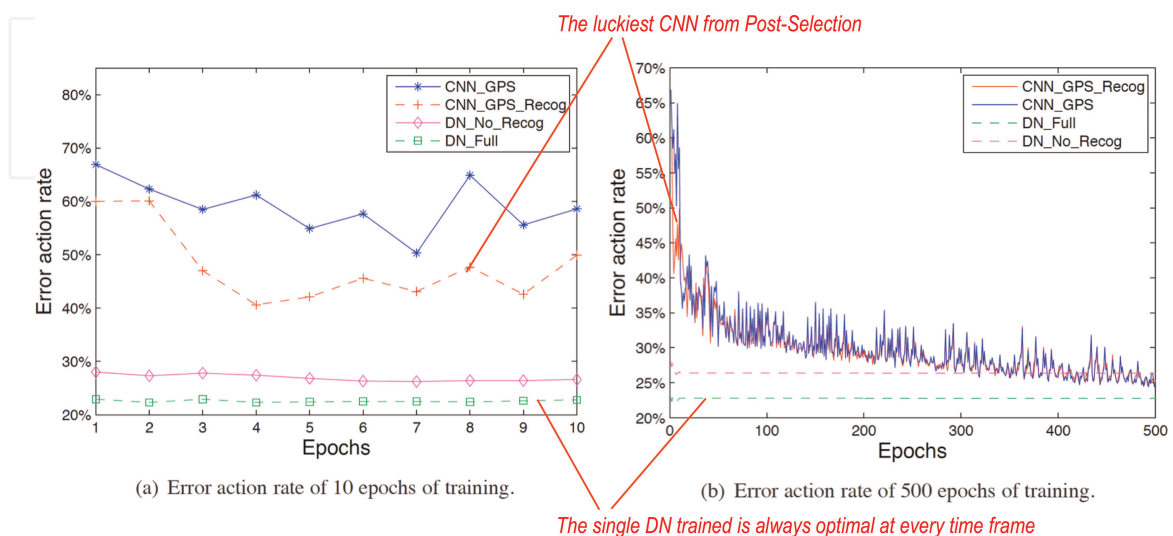
Similarly, we define *the hyper-parameter validated luckiest initial weights* from PSUVS:

$$\mathbf{w}^* = \arg \min_{1 \leq i \leq n} \frac{1}{k} \sum_{j=1}^k e_{i,j}. \quad (11)$$

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

From a statistical point of view, the initial hyperparameter 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. (8).

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 trustworthy in practice.



**Figure 3.**

Comparison between the validation error of the luckiest CNN trained by batch error-backprop and the test error of a DN across different epochs through the fit data. “Recog” and “full” means teach where-what rules. CNN shows the luckiest network on validation set only. Adapted from [86].

We also need to be aware of another protocol flaw: Random seeds and hyperparameters 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 hyperparameter vector uses a different seed. Similarly, Type-3 validation seems unnecessary with  $k = 1$  but the search for the luckiest hyperparameter vector is embedded into the search for the luckiest weights, where each random seed uses a different hyperparameter 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 to do 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 Ref. [81], the test error rate is smaller than the validation error for 7CNNs, causing a reasonable suspicion that PSUTS could be used instead of PSUVS.)

The following subsection discusses the luckiest network with the luckiest hyperparameter 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 conclude 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 luck in data partition that a typical user does not have during 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 evolutionary 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, fit set  $F$ , validation set  $V$ , and test set  $T$ , so that Eq. (6) holds true. For realistic applications, we should assume that  $F$ ,  $V$ , and  $T$  are mutually independently drawn from  $S$  so that  $F$ ,  $V$  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  $F$ ,  $V$ , and  $T$ , respectively, are mutually independent.

Using the fit set  $F$ , one trains  $nk$  networks, where  $n$  and  $k$  are the number of hyperparameter vectors  $\mathbf{a}$ 's and random weight vectors  $\mathbf{w}$ 's, respectively, using a training algorithm (e.g., error-backprop),

$$N(\mathbf{a}_i, \mathbf{w}_j) \leftarrow f_{\mathbf{a}_i, \mathbf{w}_j}(F). \quad (12)$$

This is like a teacher trains  $kn$  students in a class. The teacher knows that the fit error on  $F$  does not predict the validation error well, due to the possibility

of overfitting. One extreme example is the above nearest-neighbor classifier with distance threshold  $d = 0$ .

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 observing 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. (8). This is like the teacher colluding 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 Expected error from the luckiest architecture

Suppose that a user has bought the luckiest network architecture  $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$  and tests on his new test data  $T'$  randomly drawn from real-world  $S$ , independent of  $V$  and  $T$ . 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 on  $T'$ . We need to estimate the expected error rate of  $E(e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})|T')$  on the new test set  $T'$ .

*Theorem 1.3 (Expected error from the luckiest architecture).* Suppose that during in-house evaluation, the Type-1 order lucks and Type-2 weight lucks are simulated by casting dice as multiple tests with the luckiest errors to be  $e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}), j = 1, 2, \dots, n$ . Then, the expected error at the user end by casting dice for  $\mathbf{w}$  on a new test set  $T'$  is

$$E(e(\mathbf{a}_{i^*}, \mathbf{w})|T') = \frac{1}{n} \sum_{j=1}^n e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}|T). \quad (13)$$

instead of

$$E(e(\mathbf{a}_{i^*}, \mathbf{w})|T') = \min_{1 \leq j \leq n} e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}|T). \quad (14)$$

assuming the user's environment  $S'$  and the in-house environment  $S$  are the same.

*Proof.*  $S = S'$  means that the hyper-parameters  $\mathbf{a}_{i^*}$  from  $T$  can be inherited by  $T'$ . However, the user must cast dice for the Type-1 order lucks and Type-2 weight lucks. That is,  $\mathbf{w}$  on the left side needs to be re-estimated. Without actually conducting the training and testing at the user end, the seller can use the existing in-house tests,  $e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}|T), j = 1, 2, \dots, n$ . Without knowing the probability of each  $e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}|T)$ , it is reasonable to estimate that each is equally likely. We must not take the expected error as  $\min_{1 \leq j \leq n} e(\mathbf{a}_{i^*}, \mathbf{w}_{j^*}|T)$  because the user needs to try its luck on  $\mathbf{w}$  on the left side. The above conclusion results.  $\square$

The above theorem tells us that the error rate of the luckiest network from a single validation set in PSUVS is misleading without any data-partition validation. This is because the error rate is a random function, depending on not only many random initial weights, many hyperparameters, and local lucks of error-backprop (e.g., learning rates), but also a particular data-partition  $FUVUT$ . This seems especially true if the data  $D$  were made public and overworked during ImageNet 2010–2014 ([71], 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 fit-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.

One may argue that his luckiest number  $x$  for the architecture hyper-parameter vector is based on the validation set, the test set, or a combination of both and therefore, it is not falsely generated. However, his luckiest number  $x$  is based on post-selections from many trained systems after “seeing” their performances on the validation set, the test set, or a combination of both. As **Figure 1** illustrated, his luckiest number  $x$  along “without abstraction” cannot predict the performance during the deployment to the real world that is in fact along “with TM abstraction.”

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 [24–38], except [27], let alone whether the reported error is from the validation error  $V$  or the test set  $T$ .

### 2.10.1 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  $e_{ij}$  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*:

$$e_{i^*j^*} = \min_{1 \leq i \leq k} \min_{1 \leq j \leq n} e_{ij}. \quad (15)$$

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

Some researchers have claimed that the test data was “unseen” by trained systems when they were tested, but the network selector has seen their performances before making a selection.

Imagine that we want to remove lucks in the above expression, by using averages like we did in Eq. (9) to give *the error of the luckiest architecture with validated weights from PSUTS*:

$$\mathbf{a}_{*j^*} = \arg \min_{1 \leq j \leq n} \frac{1}{n} \sum_{i=1}^k e_{ij}. \quad (16)$$

However, the above error is still flawed since each term under minimization has peeked into test sets. Instead, it is better to use  $e_{ij}$  in Eq. (8) which does not use the test sets. Of course, the test error rate of  $e_{ij}$  in Eq. (8) tends to be larger than that from Eq. (15).

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  $F$  are not disjoint with  $T$ . If  $F = V$ , we call it validation-vanished PSUTS. If  $F = T$ , 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. (15). That is why we need computational resources and a learning experience in the Four Learning Conditions.

Although PSUVS has flaws in post-selection and a lack of three types of validation to be discussed below, the key difference between PSUVS and PSUTS does not guarantee that PSUVS reports a low error rate as PSUTS. 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 Krizhevsky & Hinton 2017 ([81], p. 88). This piece of evidence supports that Krizhevsky & Hinton probably used PSUTS instead of PSUVS.

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 authors Krizhevsky & HiHintonnton of [81] wrote: “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” ([81], p. 88), there is no evidence to rule out what he meant was the possibly “cracked” test set is not necessarily the same as the original test set.

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” ([71], footnote, p. 214). 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 (fit and validation) sets. The actual number of trained networks over random seeds and hyper-parameters was hardly reported. Krizhevsky & Hinton 2017 [81] lacks due transparency about the post-selection stage.

Geoffrey Hinton admitted in his PubPeer response to questions raised on PubPeer toward [24] that Krizhevsky & Hinton [81] reported only the “luckiest” network. They did not say that their post-selections used the validation set, the test set, or both.

Here is a piece of direct evidence: Table 2 of Krizhevsky & Hinton [81] copied to **Figure 4**. In the last row “7 CNNs,” the test error 15.3% (100 images per class) is smaller than the validation error 15.4% (50 images per class). ImageNet provided on average 1000 fit images for each test set; 1000 classes amounts to a little over  $1000 \times 1000 = 1,000,000$  images in the fit set. Note the authors claimed the 7 CNNs were “pre-trained” to classify the entire ImageNet 2011 Fall release. The ILSVRC-2011 fit set could be a subset of the ILSVRC-2012 fit set (increased 4.2%). The  $100 \times 1000 = 100,000$  test images may change from 2011 to 2012 [[71], Tables 2 and 3] but the 1000 test-class labels should remain the same from 2011 to 2012. Thus, the luckiest network from post-selections for ILSVRC-2011 (e.g., that minimizes the sum of fit error, validation error, and test error) should be almost the luckiest for ILSVRC-2012 as only 4.2% images were added into ILSVRC-2012.

**Table 2. Comparison of error rates on ILSVRC-2012 validation and test sets.**

Model	Top-1 (val, %)	Top-5 (val, %)	Top-5 (test, %)
<i>SIFT + FVs<sup>6</sup></i>	–	–	26.2
1 CNN	40.7	18.2	–
5 CNNs	38.1	16.4	<b>16.4</b>
1 CNN*	39.0	16.6	–
7 CNNs*	36.7	15.4	<b>15.3</b>

In *italics* are best results achieved by others. Models with an "\*" were "pre-trained" to classify the entire ImageNet 2011 Fall release (see Section 7 for details).

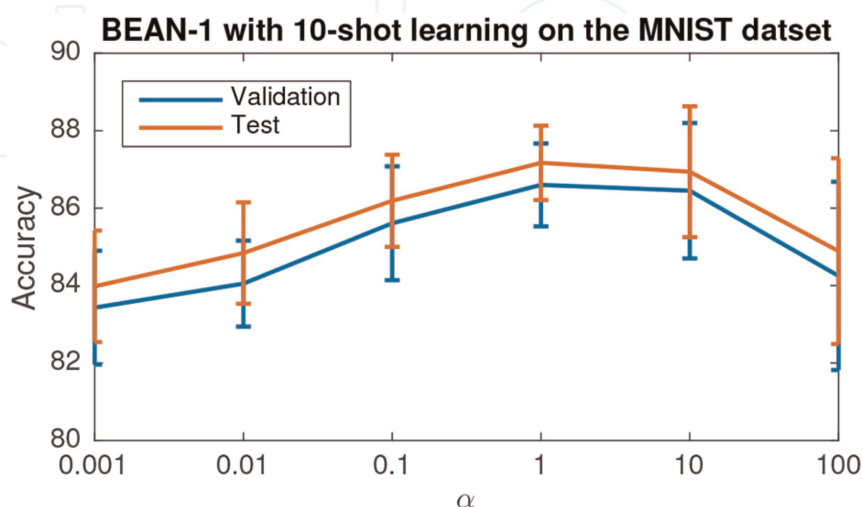
**Figure 4.**

Table 2 from Krizhevsky & Hinton 2017 [81]. In the last row, the test error 15.3% is smaller than the validation error 15.4%. The ImageNet-reported numbers for the second and third bold numbers along the Top-5 (test %) column are 16.422 and 15.315, respectively.

Imagine the following questions in a civil court that should be answered based on a *preponderance of evidence*:

1. Did the authors conduct post-selections for Ref. [81] using a fit set, a validation set, a test set, or a combination thereof?
2. As the validation error 15.4% is larger than the test error 15.3%, is it more likely that the authors used a test set (or both test and validation sets) for **Figure 4** than not using a test set at all?
3. Did the authors lack due transparency about the post-selection stage in Ref. [81]?

For examples of PSUTS by other authors, see **Figure 5** from [Figure 7, [87]], error-backprop consistently results in lower validation accuracies than the test accuracies



**Figure 5.**

The average  $\bar{e}$  and the standard deviation of  $\{e_i\}$  for different values of a regularization hyperparameter  $\alpha$ . PSUTS was probably used instead of PSUVS because the test data are better than the validation data like Krizhevsky & Hinton 2017 ([81], p. 88). Courtesy of Ref. [87].

(about 0.5% lower compared to about 0.1% lower in [81]). Is there other evidence of using PSUTS instead of PSUVS, similar to [81]? The availability of test sets to the programmers in a project seems to be indeed addictive toward PSUTS, away from PSUVS. The standard deviation around 1% is clashed with our Theorem 1.5. Our experience with our experiments with error-backprop training for CNN indicated that the maximum and the minimum values of the distribution of fit accuracies are drastically different for different random seeds, with fit accuracies spreading uniformly between 20 and 90%. Section 1.3 will discuss why. If Theorem 1.5 in Section 1.3 is correct, the deviation bars seem too small and the 20 runs in Figure 7 of [87] could be the best 20 among many more random seeds the programmer has tried. We hope that the authors provide the source program.

### 2.10.2 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 error, fit, 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 fit errors, and validation errors, and test errors, respectively, not just the standard deviation in **Figure 5**. 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 overfitting to the validation/test set. Unfortunately, none of Refs. [23–25, 27, 29, 31, 66, 76] reported such distribution characteristics other than the minimum value  $e_{i^*, j^*}$ .

Furthermore, such use of test sets to post-select networks resembles hiring a larger number  $kn$  of random test takers and reporting 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 purposes 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 fit set  $F$ ; 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  $e_{i^*, j^*}$ ,

$$e'_{i^*, j^*} \gg e_{i^*, j^*} \quad (17)$$

because Eq. (15) 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 human learning involves learning invariant rules. The emergent universal Turing machine in Section 1.4.4 is meant to learn such invariant rules as explained in Ref. [78], which are more powerful than the (greedy) data-fitting methodology of so-called Deep Learning in AI. A post-selection is a technically flawed protocol that hides the key weakness of this data-fitting methodology.

PSUTS is tempting especially when test sets are available to the authors of a paper.

Weng [88, 89] argued that the claims by some public speakers that such misleading errors have approached or even succeeded human performance [71] that are controversial since there are no explicit competition rules that ban test sets to be used for post-selections.

## 2.11 Human PSUTS

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

*Definition 1.6 (Human PSUTS).* After planning experiments or knowing what will be in the fit set  $F$  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 designs a micro-world and collects 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 a programmer has in hand picking features.

Poggio et al. [90] 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 toward [19] on PubPeer were not answered by the authors.

To understand why post-selections are misconduct that gives misleading results, let us derive the following important theorem.

*Theorem 1.4 (PGNN Supremacy).* Given any validation error rate  $e_v \geq 0$  and test error rate  $e_t \geq 0$ , using Post-Selections the PGNN classifier satisfies any required  $e_v$  and  $e_t$ , if the author is in the possession of the test set  $P = V \cup T$  and both the storage space and the time spent on the Post-Selections are finite but unbounded if the Post-Selection is allowed.

*Proof.* Because the number of seeds to be tried during the post-selection is finite but unbounded, we can prove that there is a finite time at which a lucky seed  $s$  will produce a good enough verification error and test error. Although the waiting time is long, the time is finite because  $V$  and  $T$  are finite. Let us formally prove this. Suppose  $l$  is the number of labels in the output set  $L$ . For the set of queries in  $V$  and  $T$ , there are  $k$  (constant) outputs that must be guessed. The probability for a single guess to be correct is  $1/l_0$ ,  $l_0 = \|L\|$ , due to the uniform guess in  $L$ . The probability for  $k$  guesses to be all correct is  $(1/l_0)^k = 1/l_0^k$  because guesses are all mutually independent. The probability of guessing at least one label wrong is  $1 - 1/l_0^k$ , with  $0 < 1 - 1/l_0^k < 1$ . The probability for as many as  $n$  runs of Post, all of which do not satisfy the  $e_v = 0$  and  $e_t = 0$ , is

$$p(n) = \left(1 - 1/l_0^k\right)^n \rightarrow 0, \quad (18)$$

as  $n$  approaches infinity, because  $0 < 1 - 1/l_0^k < 1$ . Therefore, within a finite period, a process of trying incrementally more networks using Post will get a lucky network that satisfies both the required  $e_v$  and  $e_t$ . This is the luckiest network from the post-selection. □



Theorem 1.4 has established that post-selections can even produce a superior classifier that gives any required validation error and any test error, including zero-value requirements! Yes, while the test sets are in the possession of authors, the authors could show any superficially impressive validation error rates and test error rates (including even zeros!) because they used post-selections without a limit on resources (to store all data sets and to search for the luckiest network). It is of course time-consuming for a program to search for a network whose guessed labels are good enough. But such a lucky network will eventually come within a finite time frame!

Yes, while the test sets are in the possession of authors, the authors could show superficially impressive validation error rates and test error rates because they used post-selections without a limit on resources (to store all data sets and to search for the luckiest network).

Theorem 1.4 has established that post-selections can even produce a perfect classifier that gives a zero validation error and zero test error under the flawed protocol to mislead us!

The above theorem means any competitions without an explicit limit on storage and time spent on post-selections are meaningless, like ImageNet and many other competitions. It is of course time-consuming for a program to search for a network whose guessed labels are good enough.

Corollary 1.1 (Misconduct papers). *If a paper trains more than one system, the performance data from the paper are misleading if the paper does not report the number of systems trained in Post-Selections, the amount of computational resources (i.e., the amount of storage, the number of computers, the type of computers, and the network refreshing rates), the amount of waiting time, the fit errors, the validation errors, and the test errors of all trained systems.*

*Proof.* From Theorem 1.4, if post-selections are allowed, a nearest-neighbor classifier with a random distance threshold can satisfy any nonzero validation error and any nonzero testing error using post-selections, since the training set, validation set and test set are all in the possession of the authors.  $\square$

This corollary is a scientific basis for the author to raise a violation of protocols and a lack of transparency about the post-selection stage in almost all machine learning papers that appeared in *Nature*, *Science*, *Communication of ACM* [91] and other publication venues since around 2015. Since all the fit sets, validation sets, and test sets are in the possession of the authors, many papers have claimed misleading results using a flawed protocol.

### 3. Why error-backprop needs post-selection

This section discusses a global view, which is new as far as the author is aware, about why error-backdrop 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 also from the following discussion, we will concentrate on batch learning mode. Namely, we let the network “see” the entire fit set  $F$  for each network update.

Let us first consider a well-known neuronal model that applies 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:

$$\phi\left(\sum_{i=1}^n w_{ij}y_i\right) = z_j \quad (19)$$

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

$$\eta(y_1, y_2, \dots, y_n) \triangleq \eta \mathbf{y} \quad (20)$$

where  $\eta$  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

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + w_2 \mathbf{y} \quad (21)$$

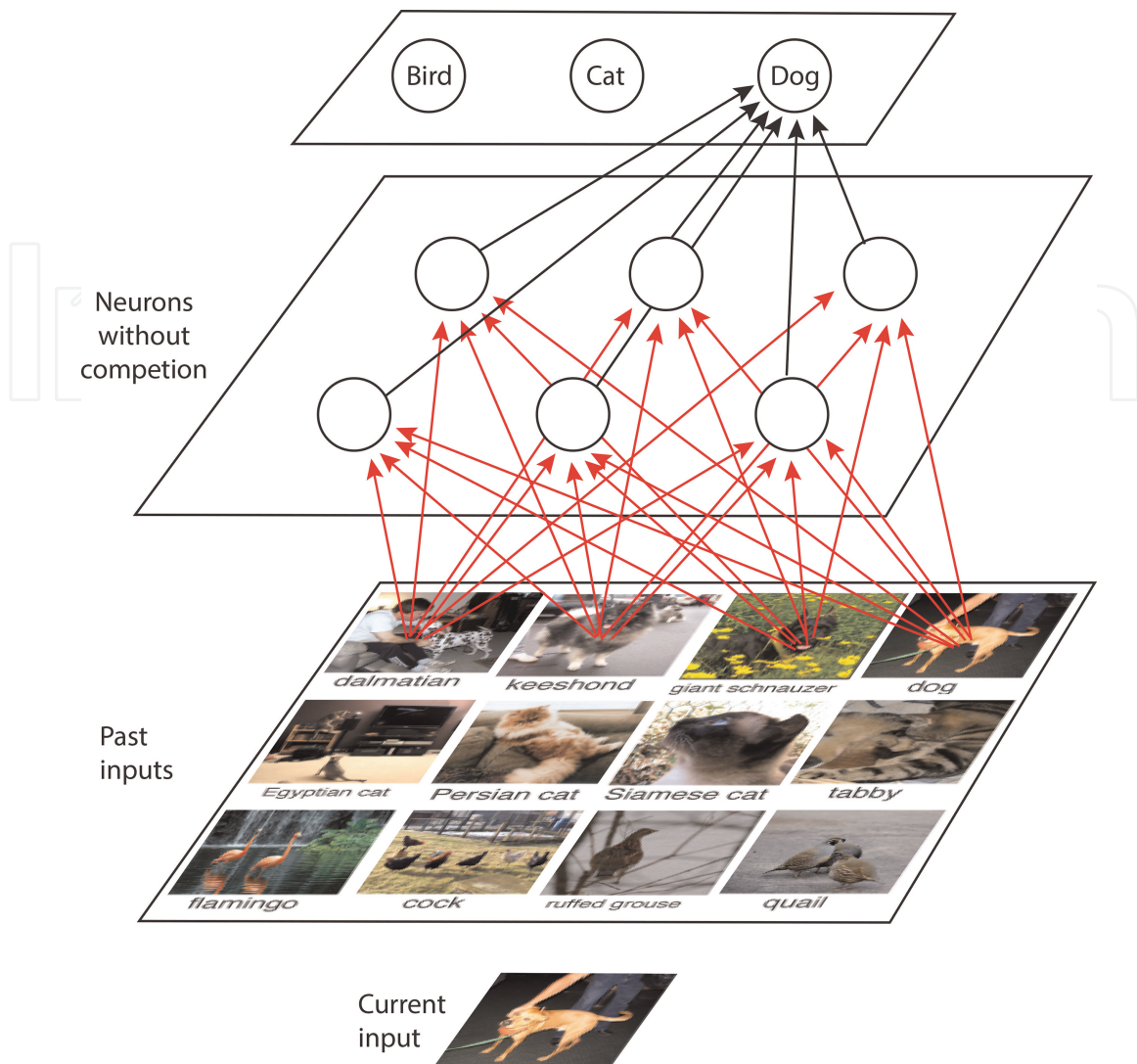
where  $w_2$  is the learning rate. We use the  $w_2$  to relate better the optimal Hebbian learning, called LCA, used by DN in Section 1.4. At this point, the following theorem is in order.

*Theorem 1.5 (Lacks of error-backprop). Error-backprop lacks (1) energy conservation, (2) an age-dependent learning rate, and (3) competition-based role-determination.*

*Proof.* Proof of (1): If pre-synaptic input vectors  $\{\mathbf{y}\}$  are similar, multiple applications of Eq. (21) 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 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 1.4. 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. (21) 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 weights using their gradients. Namely, there is no competition-based role-determination in error-backprop.  $\square$

The meaning (3) of Theorem 1.5 are illustrated by **Figure 6**. 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 the input area 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 the 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 [77], and 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.

It is important to note that the key point is global role-determination through global competition instead of local inhibition (such as pooling for resolution-reduction [16]) or attention because they do not enable role-determination among neurons in a global way.



**Figure 6.** 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 Ref. [71].

**Theorem 1.6 (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 fit 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. (21) 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 **Figure 6** does not matter, 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 fit error rate, the validation error rate, and the test error rate, because gradient updates are local and inherit such initial roles. □

**Theorem 1.7 (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 fire, 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 which 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% back and 50% white). Each neuron  $z$  in area  $A_l$  is abstract which means that its firing means an abstract class  $C$  that  $\mathbf{x}$  belongs to, with  $\bar{p}(z)$  being small corresponds to top-1 among  $n$  neurons. Then, the CNN must 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 **Figure 6**, 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 1.5 through 1.7 and their proofs, we can see that the lack 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 in selecting 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 overfit, e.g., variance-based stochastic gradient decent [92], saddle-free deep network [93], drop out [94], implicit regularization during gradient flow [95]. There have been other papers [96, 97] that weakly discussed the local minima issue. The main weakness is that these papers all address only local issues of neural networks and do not mention post-selections. There have been no papers, as far as the author is aware, that have correctly established the freedom from local minima with gradient-based learning techniques for neural networks.

In contrast, the optimality work here is a global method for a single trained network without 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.

From the above discussion, we can obtain a more general observation: Any locally greedy method is always a local minimum, due to a lack of competition combined with the restrictions of the Four Learning Conditions. This more general observation has great importance to not only neural networks, but also to other learning modes, including reinforcement, swarm, reservoir, and evolutionary learning modes.

This observation should also apply to social sciences, such as governing, national development, international relations, and world peace. Namely, social science must consider competitions and the Four Learning Conditions for governing, national development, international relations, and world peace.

The following post-selection-free method is restricted by the Four Learning Conditions.

#### 4. How a DN avoids post-selections

A brain does not use post-selection at all, whether PSUVS or PSUTS, because every human child must develop in a human environment to make his living. He should not be covered up and not reported, regardless of how well or badly he performs.

Cresceptron in the 1990s [16, 43–46] and later DN [47, 57–62] 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 Four Learning Conditions.

##### 4.1 New AI metrics: Developmental errors

In contrast to post-selections likely used by Refs. [14, 19, 21, 23–25, 27, 29, 31, 66, 76, 90] including probably AlphaGo [26], AlphaGo Zero [28], AlphaZero [65], AlphaFold [30] and MuZero [67] and many others, we define and reported developmental errors that include all errors occurred through lifetime of each learning network:

**Definition 1.7 (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 weighs, 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 a 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:*

$$(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{a}_t) = f(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1}, \mathbf{a}_{t-1}) \quad (22)$$

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 an error, the error should be treated as the 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

$$\bar{e}(t) \triangleq \frac{1}{t} \sum_{i=0}^t e_i, \quad (23)$$

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

$$\bar{e}(t) = \frac{t-1}{t} \bar{e}(t-1) + \frac{1}{t} e_t. \quad (24)$$

Namely, all errors across a lifetime, at every time instance, are caught by the developmental error. 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.

It is important to note that developmental error is recursively weighted by “reinforcement value” as explained in Refs. [42, 75], not an equal average over time. For example, driving across a red light occurs only a few times in a life, but has a high actual *serotonin* punishment and recalled cognitive punishment [42, 75] through communicative learning (i.e., verbal communications without experiencing a death and without any time-discount like in Q-learning, but hearing horrible stories and trusted people warn against it). A DN further models unified scaffolding [79]: Adults avoid children’s behaviors.

An optimal network that gives the lowest possible developmental error, among all possible networks under the same Four 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 Four Learning Conditions because the maximum-likelihood optimality should give equivalent networks of the same developmental error.

However, in practice, the learning experience in the Four 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 robots 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 [72]), the winner is unlikely to be one that uses a brute force method but has an excessive amount of computational resources and manpower. ImageNet competitions [71] are flowed also in this sense.

Although not formally defined as developmental errors, Cresceptron [16] and Developmental Networks [41, 98, 99] reported developmental errors.

Namely, the developmental error, unless stated otherwise for a particular period, is the average lifetime error from inception. To report more detailed information about the process of developmental errors  $\{e_t \mid t \geq 0\}$ , 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 specific periods, 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

$$\mathbf{c}_t \triangleq (\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \quad (25)$$

maybe significantly different from the nearest matched learned weight vectors of all hidden neurons. If that happens and there are still new hidden neurons that have not fired, a free-state neuron that happens to be the best match is spawned that perfectly memorizes this new context regardless of 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 [98], 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 the 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.

## 4.2 Neuronal competitions

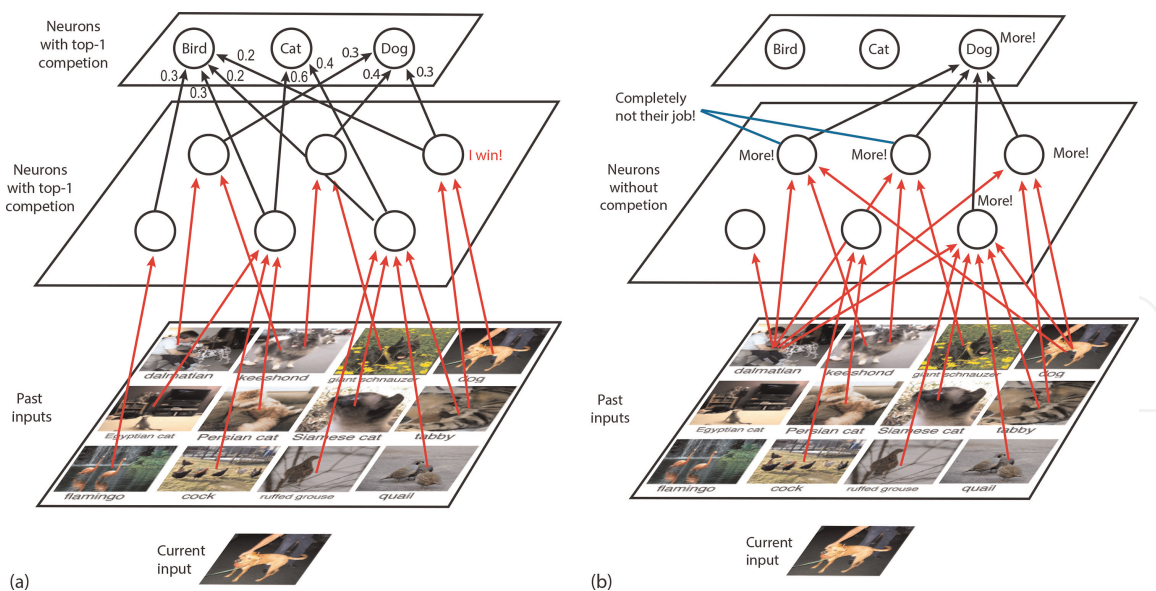
As discussed above, error-backprop learning is without neuronal competition. The main purpose of competition is to automatically assign roles to hidden neurons. Below, we consider two kinds of Convolution Neural Networks (CNNs), sensory networks and sensorimotor networks. A sensory network is feedforward, 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.

### 4.2.1 Sensory networks

Let us first consider the case of feed-forward networks as illustrated in **Figure 7**. **Figure 7(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 updating the network incrementally.

Every layer in Cresceptron consists of an 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 reduced from the lower layer to the higher layer was called resolution reduction (also called drop-out).



**Figure 7.** How competition automatically assigns roles among hidden neurons without a central controller: The case for automatically constructing 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 Ref. [71].

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  $\mathbf{x} \in X$  arrive sequentially, the top-1 competition in the hidden layer in **Figure 7(a)** enables each hidden neuron to respond to multiple features, indicated by the typically multiple upward arrows, one from each image, pointing to a hidden neuron. This amounts to incremental clustering based on top- $k$  competition. The weight vector of each hidden  $Y$  neuron corresponds to a cluster in the  $X$  space. In **Figure 7(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 in 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 [77] 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 of 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.

$$\mathbf{v} \leftarrow \left(1 - \frac{1}{a}\right)\mathbf{v} + \frac{1}{a}r\mathbf{x}. \quad (26)$$



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 dual optimality resolves the three problems in Theorem 1.5.

There have been many ideas about “adaptive learning rate” (e.g., adaptive momentum), but they all do not have the dual-optimality of LCA that requires the age-dependent learning rate. For more details and a mathematical derivation, see Ref. [77].

**Figure 7(b)** shows how the three neurons in the  $Z$  area update their weights so that the weight from the second area to the third area becomes 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 1.6.

However, the 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.

#### 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 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 **Figure 7**, but also a sensorimotor feature. By sensorimotor feature, we mean that the firing of each hidden neuron in **Figure 7** is determined not just by the current image  $\sigma$  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.

### 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 the function  $\delta : Q \times \Sigma \mapsto Q$ , where  $\Sigma$  is the set of input symbols and  $Q$  is the set of states. Each transition is represented by

$$(q, \sigma) \xrightarrow{f} q' \quad (27)$$

*AFA as a control of any Turing machine* Weng [41] extended the definition of the FA so that it outputs its state so the resulting FA becomes an Agent FA (AFA). Further, Weng [41] extended the action  $q$  to the machinery of the Turing machine (see **Figure 8**) 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 [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;  $\sigma$  is the bottom-up sensory input to the same neuron. If  $\delta$  has  $n$  transitions,  $n$  hidden neurons in the  $Y$  area are sufficient to memorize all the transitions that are observed sequentially, one transition at a time.

We should not use symbols like  $\sigma$  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:

$$\begin{bmatrix} Z(0) \\ Y(0) \\ X(0) \end{bmatrix} \rightarrow \begin{bmatrix} Z(1) \\ Y(1) \\ X(1) \end{bmatrix} \rightarrow \begin{bmatrix} Z(2) \\ Y(2) \\ X(2) \end{bmatrix} \rightarrow \dots \quad (28)$$

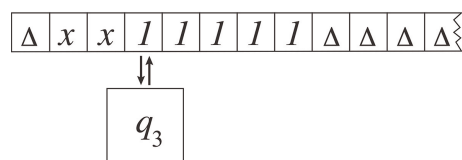
where  $\rightarrow$  means neurons on the right-hand side use the input neurons on the left-hand side and mutually compete (among connected neurons on the right-hand side) to fire, as explained below, without iterations. Namely, by unfolding time, the spatially recurrent DN becomes non-recurrent in a time-unfolded and time-sampled DN. With the LCA update, Weng [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.

#### 4.4 DN as a ML-optimal universal Super-Turing machine

The traditional Turing Machine (TM) is a human handcrafted machine, as illustrated in **Figure 8**.

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 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 [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!



**Figure 8.**  
 A Turing machine has a tape, a read-write head, and a transition function with a current state.

#### 4.5 DN 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 1.2. This means that a DN ML optimally learns Autonomous Programming for General Purposes (APFGP) [58, 100]. Based on the capability of APFGP, Weng argued that APFGP is a characterization of conscious machines [59–62] 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 that APFGP is not a vague idea and how APFGP by DN avoids post-selections.

#### 4.6 DN-2 algorithm

Let us go through the DN-2 algorithm here so that we can see that DN is fully detailed 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 [16, 19, 23–25, 27, 29, 31, 43–46, 66, 76, 90]. 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 1.2.

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:

$$(\mathbf{y}', M'_Y) = f_Y(\mathbf{c}_Y, M_Y) \quad (29)$$

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 [75, 77]. 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:

$$(\mathbf{z}', M'_Z) = f_Z(\mathbf{c}_Y, M_Z) \quad (30)$$

where  $f_Z$  is the  $Z$  area function using LCA [75, 77] 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. (29) and area function  $f_Z$  in Eq. (30) 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.

In a DN-1, each neuron does not have its competition zone. All neurons in an area share the same competition zone. A neuron files if and only if its pre-action potential is at the top- $k$  among the pre-action potentials of its competing neurons.

In a DN-2, each neuron has its competition zone, so that the neuron files if and only if its pre-action potential is at the top- $k$  among the pre-action potentials of its own competing neurons.

Regardless of DN-1 or DN-2, the freedom from post-selection discussed below applies. The ML optimization of DN-1 and DN-2 is rooted in the optimality of LCA and extends to the entire network and entire lifetime.

#### 4.7 Maximum likelihood needs no post-selections

Given the Four 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 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  $\theta$  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  $\theta$  corresponds to the  $\theta$  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:

$$(\theta^*, \mathbf{y}^*, \mathbf{z}^*) = \arg \max_{(\theta, \mathbf{y}, \mathbf{z})} f_\theta(\mathbf{x}, \mathbf{z}). \quad (31)$$

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  $\theta_t^*$  is computed in a closed-form by the incremental version of Eq. (31) where  $f$  uses context  $\mathbf{c}_{t-1} = (\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_{t-1})$ :

$$(\theta_t^*, \mathbf{y}_t^*, \mathbf{z}_t^*) = \arg \max_{(\theta_t, \mathbf{y}_t, \mathbf{z}_t)} f_{\theta_t}(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}^*, \mathbf{z}_{t-1}^*). \quad (32)$$

The DN computes the above expression for each time  $t$  in a closed form without conducting any iterations [41, 98].

How about initial weights? Inside  $\theta$ , 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. (26) shows that the initial weights of this free neuron are 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 [98] 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 of 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.

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. [98] proved the ML-optimality for DN-2. Since the number of transitions 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 1.2. 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 of the Four Learning Conditions in Definition 1.2, a DN is not 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.

#### 4.9 Comparison with HMM

It is important to compare the traditional Hidden Markov Model (HMM) [101] 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 likes. (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).

## 5. Experiments

### 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 [102], (2) audition to learn phonemes with a simulated cochlea and the corresponding behaviors [103], (3) acquisition of English and French in an interactive bilingual environment [104], 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) [105]. 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 Four Learning 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, **Figure 3** 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, **Figure 3** 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 needs many epochs to reduce its errors and only very slowly. At the end of the 500th epoch, the error of the luckiest CNN trained by error-backprop is still considerably

higher than the full DN. Furthermore, as shown in **Figure 3**, teaching invariant concepts, i.e., abstraction in Theorem 1.2, are used for reducing the optimal errors. For more detail, the reader is referred to Ref. [86].

### 5.3 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 language acquisition, while the engine learns in “lifetime.” Although all the teams were 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  $\mathbf{x}$  be the image at each time instance and  $\mathbf{z}$  be the pattern of landmark location-and-type and action of navigation, the DN became a vision-guided navigation machine. When we let  $\mathbf{x}$  be the frame of the firing pattern of hair cells in the cochlea at each time instance and  $\mathbf{z}$  be the dense states/ contexts and the sparse type of sounds, the DN became an auditory-recognizer machine. When we let  $\mathbf{x}$  be a time frame of the vector of the word (either English or French) and  $\mathbf{z}$  be the language kind (neutral, English, and French) and the 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 Four 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 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 fit 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 [71] 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 to exist as APFGP robots. Hopefully, as an APFGP platform, this new kind of human-wearable robot will be useful for practitioners to produce various kinds of intelligent auto-programmed software. The author predicts that such a new kind of AI system will considerably alleviate the high brittleness of traditional AI software and traditional robot software in the 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 relatively 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 goal but have not reached this experimental goal yet.

## 6. Conclusions

We used intuitive terms but formal ways to discuss post-selections. Post-selections suffer from two types of misconduct, cheating and hiding. They are like casting a dice when one purchases a product. Whether the product works as the ad data have claimed depends on the outcome of casting a dice. This is an unacceptable uncertainty for many AI applications, e.g., a driverless car. The public and media have gained the 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 [71, A2, p242]) and the work is laudable. However, this chapter 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 Four Learning Conditions. Some researchers have raised that it seems that the winning team is the one that has more computational resources and manpower at its disposal. The new developmental error metrics under the Four Learning Conditions hopefully encourage future AI competitions to compare methods under the same Four Learning Conditions. Considering DN, as a much-simplified model for conscious learning in an animal, it seems not baseless to guess that each biological brain is probably ML-optimal (of course in a much richer sense) across a lifetime, e.g., due to the pressure to compete at every age. The Four 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 non-scalable 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 of how large their apparent sizes are.


### Author details

Juyang Weng  
Brain-Mind Institute and GENISAMA, USA

\*Address all correspondence to: [juyang.weng@gmail.com](mailto:juyang.weng@gmail.com)

### IntechOpen

---

© 2023 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 



## References

- [1] Montfort N. *Twisty Little Passages: An Approach to Interactive Fiction*. Cambridge, MA: MIT Press; 2005
- [2] Turing AM. Computing machinery and intelligence. *Mind*. 1950;**59**:433-460
- [3] Weng J. Symbolic models and emergent models: A review. *IEEE Transactions on Autonomous Mental Development*. 2012;**4**(1):29-53
- [4] Russell S, Norvig P. *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River, New Jersey: Prentice-Hall; 2010
- [5] Minsky M. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*. 1991;**12**(2): 34-51
- [6] Lenat DB, Miller G, Yokoi TT. CYC, WordNet, and EDR: Critiques and responses. *Communications of the ACM*. 1995;**38**(11):45-48
- [7] Gomes L. Machine-learning maestro Michael Jordan on the delusions of big data and other huge engineering efforts. *IEEE Spectrum*. 2014
- [8] Rumelhart DE, McClelland JL, the PDP Research Group. *Parallel Distributed Processing*. Vol. 1. Cambridge, Massachusetts: MIT Press; 1986
- [9] McClelland JL, Rumelhart DE, The PDP Research Group, editors. *Parallel Distributed Processing*. Vol. 2. Cambridge, Massachusetts: MIT Press; 1986
- [10] Krishna R, Zhu Y, Groth O, Johnson J, Hata K, Kravitz J, et al. Visual genome. *Int'l Journal of Computer Vision*. 2017;**123**(1):32-73
- [11] Funahashi KI. On the approximate realization of continuous mappings by neural networks. *Neural Networks*. 1989; **2**(2):183-192
- [12] Poggio T, Girosi F. Networks for approximation and learning. *Proceedings of the IEEE*. 1990;**78**(9): 1481-1497
- [13] Kohonen T. *Self-Organizing Maps*. 3rd ed. Berlin: Springer-Verlag; 2001
- [14] Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. 1980;**36**:193-202
- [15] Oja M, Kaski S, Kohonen T. Bibliography self-organizing maps (som) papers: 1998-2001 addendum. *Neural Computing Surveys*. 2003;**3**:1-156
- [16] Weng J, Ahuja N, Huang TS. Learning recognition and segmentation using the Cresceptron. *Int'l Journal of Computer Vision*. 1997;**25**(2):109-143
- [17] Fukushima K, Miyake S, Ito T. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*. 1983; **13**(5):826-834
- [18] Serre T, Poggio T, Riesenhuber M, Wolf L, Bileschi S. High-performance vision system exploiting key features of visual cortex. US Patent, US7606777B2 [Accessed: Sept. 1, 2006]
- [19] Fei-Fei L, Fergus R, Perona P. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2006;**28**(4): 594-611

- [20] Weng J. Dialog initiation: Modeling AMD: Closed skull or not? IEEE CIS Autonomous Mental Development Newsletter. 2012;9(2):10-11
- [21] Werbos PJ. The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting. Chichester: Wiley; 1994
- [22] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of IEEE. 1998;86(11): 2278-2324
- [23] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. Vol. 25. Cambridge, Massachusetts: MIT Press; 2012. pp. 1106-1114
- [24] LeCun Y, Bengio L, Hinton G. Deep learning. Nature. 2015;521:436-444
- [25] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature. 2015; 518:529-533
- [26] Silver D, Huang A, Hassabis D, et al. Mastering the game of go with deep neural networks and tree search. Nature. 2016;529:484-489
- [27] Graves A, Wayne G, Reynolds M, Hassabis D, et al. Hybrid computing using a neural network with dynamic external memory. Nature. 2016;538: 471-476
- [28] Silver D, Schrittwieser J, Hassabis D, et al. Mastering the game of go without human knowledge. Nature. 2017;550: 354-359
- [29] McKinney SM, Sieniek M, Godbole V, Shetty S, et al. Int'l evaluation of an AI system for breast cancer screening. Nature. 2020;577: 89-94
- [30] Senior AW, Evans R, Hassabis D, et al. Improved protein structure prediction using potentials from deep learning. Nature. 2020;577:706-710
- [31] Bellemare MG, Candido S, Wang Z, et al. Autonomous navigation of stratospheric balloons using reinforcement learning. Nature. 2020; 588(7836):77-82
- [32] Ecoffet A, Huizinga J, Lehman J, Stanley KO, Clune J. First return, then explore. Nature. 2021;590(7847): 580-586
- [33] Saggio V, Asenbeck BE, Walther P, et al. Experimental quantum speed-up in reinforcement learning agents. Nature. 2021;591(7849):229-233
- [34] Willett FR, Avansino DT, Shenoy KV, et al. High-performance brain-to-text communication via handwriting. Nature. 2021;593(7858): 249-254
- [35] Slonim N, Bilu Y, Alzate C, Aharonov R, et al. An autonomous debating system. Nature. 2021; 591(7850):379-384
- [36] Mirhoseini A, Goldie A, Yazgan M, Dean J, et al. A graph placement methodology for fast chip design. Nature. 2021;594(7862):207-212
- [37] Lu MY, Chen TY, Mahmood F, et al. AI-based pathology predicts origins for cancers of unknown primary. Nature. 2021;594(7861):106-110
- [38] Warnat-Herresthal S, Schultze H, Shastry KL, Schultze JL, et al. Swarm learning for decentralized and

confidential clinical machine learning. *Nature*. 2021;**594**(7862):265-270

[39] Weng J, McClelland J, Pentland A, Sporns O, Stockman I, Sur M, et al. Autonomous mental development by robots and animals. *Science*. 2001; **291**(5504):599-600

[40] McClelland JL, Plunkett K, Weng J. Guest editorial: Convergent approaches to the understanding of autonomous mental development. *IEEE Transactions on Evolutionary Computation*. 2007; **11**(2):133-136

[41] Weng J. Brain as an emergent finite automaton: A theory and three theorems. *Int'l Journal of Intelligence Science*. 2015;**5**(2):112-131

[42] Wang D, Duan Y, Weng J. Motivated optimal developmental learning for sequential tasks without using rigid time-discounts. *IEEE Transactions on Neural Networks and Learning Systems*. 2018;**29**:164-175

[43] Weng J, Ahuja N, Huang TS. Cresceptron: A self-organizing neural network which grows adaptively. In: *Proc. Int'l Joint Conference on Neural Networks*. Vol. 1. Baltimore, Maryland NJ: IEEE Press; Jun 1992. pp. 576-581

[44] Weng J, Ahuja N, Huang TS. Learning recognition and segmentation of 3-D objects from 2-D images. In: *Proc. IEEE 4th Int'l Conf. Computer Vision*. New York, NY: IEEE Press; May 1993. pp. 121-128

[45] Weng J. Life is science (35): Did Turing awards go to plagiarism? Facebook blog [Accessed: March 4, 2020]. Available from: [www.facebook.com/juyang.weng/posts/10158305658699783](https://www.facebook.com/juyang.weng/posts/10158305658699783)

[46] Weng J. Did Turing awards go to plagiarism? YouTube video [Accessed:

May 27, 2020] 1:05 hours.

Available from: <https://youtu.be/EAhkH79TKFU>

[47] Weng J. Why have we passed neural networks do not abstract well? *Natural Intelligence: The INNS Magazine*. 2011; **1**(1):13-22

[48] 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*, Monterey, CA, Aug. 9–12, 2008. Monterey, CA, NJ: IEEE Press. pp. 61-66

[49] 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 Int'l Conference on Development and Learning and on Epigenetic Robotics*, Providence, RI, August 13–16 2015. NJ: IEEE Press. pp. 1-6

[50] Super CM. Environmental effects on motor development: A case of Africa infant precocity. *Developmental Medicine and Child Neurology*. 1976;**18**: 561-567

[51] Thoroughman KA, Taylor JA. Rapid reshaping of human motor generalization. *Journal of Neuroscience*. 2005;**25**(39):8948-8953

[52] Rizzotti G, Riggio L, Dascola I, Umiltà C. Reorienting attention across the horizontal and vertical meridians: Evidence in favor of a premotor theory of attention. *Neuropsychologia*. 1987;**25**: 31-40

[53] Moore T, Armstrong KM, Fallah M. Visuomotor origins of covert spatial attention. *Neuron*. 2003;**40**:671-683

- [54] Iverson JM. Developing language in a developing body: The relationship between motor development and language development. *Journal of Child Language*. 2010;**37**(2):229-261
- [55] Weng J, Luciw M. Brain-like emergent spatial processing. *IEEE Transactions on Autonomous Mental Development*. 2012;**4**(2):161-185
- [56] Weng J, Luciw M, Zhang Q. Brain-like temporal processing: Emergent open states. *IEEE Transactions on Autonomous Mental Development*. 2013;**5**(2):89-116
- [57] Weng J, Zheng Z, Xiang W, Castro-Garcia J. Auto-programming for general purposes: Theory and experiments. In: *Proc. Int'l Joint Conference on Neural Networks, Glasgow, UK, July 19–24 2020*. NJ: IEEE Press. pp. 1-8
- [58] Weng J. Autonomous programming for general purposes: Theory. *Int'l Journal of Huamnoid Robotics*. 2020; **17**(4):1-36
- [59] Weng J. Conscious intelligence requires developmental autonomous programming for general purposes. In: *Proc. IEEE Int. Conf. On Dev. Learning and Epigenetic Robotics, Valparaiso, Chile, Oct. 26–27 2020*. NJ: IEEE Press. pp. 1-7
- [60] Weng J. An algorithmic theory of conscious learning. In: *2022 3rd Int'l Conf. on Artificial Intelligence in Electronics Engineering, Bangkok, Thailand, Jan. 11–13 2022*. NY: ACM Press. pp. 1-10
- [61] Weng J. 20 million-dollar problems for any brain models and a holistic solution: Conscious learning. In: *Proc. Int'l Joint Conference on Neural Networks, Padua, Italy, July 18–23 2022*. NJ: IEEE Press. pp. 1-9. Available from: <http://www.cse.msu.edu/weng/research/20M-IJCNN2022rvsd-cite.pdf>
- [62] Weng J. A protocol for testing conscious learning robots. In: *Proc. Int'l Joint Conference on Neural Networks, Queensland, Australia, June 23 2023*. NJ: IEEE Press. pp. 1-8
- [63] Roy A. Connectionism, controllers, and a brain theory. *IEEE Transactions on System, Man, and Cybernetics—Part A; Systems and Humans*. 2008;**38**(6): 1434-1441
- [64] Felleman DJ, Van Essen DC. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*. 1991;**1**:1-47
- [65] Silver D, Hubert T, Hassabis D, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*. 2018; **362**(6419):1140-1144
- [66] Moravcik M, Schmid M, Burch N, Bowling M, et al. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*. 2017;**356**:508-513
- [67] Schrittwieser J, Antonoglou I, Silver D, et al. Mastering Atari, go, chess and shogi by planning with a learned model. *Science*. 2020;**588**(7839):604-609
- [68] Weng J. On post selections using test sets (PSUTS) in AI. In: *Proc. Int'l Joint Conference on Neural Networks, Shenzhen, China, July 18–22 2021*. NJ: IEEE Press. pp. 1-8
- [69] Weng J. A developmental method that computes optimal networks without post-selections. In: *Proc. IEEE Int'l Conference on Development and Learning, Beijing, China, August 23–26 2021*. NJ: IEEE Press. pp. 1-6

- [70] Lee H, Grosse R, Ranganath R, Ng AY. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Proc. 26th Int'l Conf. On Machine Learning, Montreal, Canada, June 14–18, 2009. pp. 609-616
- [71] Russakovsky O, Deng J, Fei-Fei L, et al. ImageNet large scale visual recognition challenge. *Int'l Journal of Computer Vision*. 2015;115:211-252
- [72] Weng J, Zheng Z, Wu X, Castro-Garcia J, Zhu S, Guo Q, et al. Emergent Turing machines and operating systems for brain-like auto-programming for general purposes. In: Proc. AAAI 2018 Fall Symposium: Gathering for AI and Natural Systems, Arlington, Virginia, October 18–20 2018. DC: AAAI Press. pp. 1-7
- [73] Ballard DH, Brown CM. *Computer Vision*. New Jersey: Prentice-Hall; 1982
- [74] Shapiro L, Stockman G. *Computer Vision*. New York: Addison-Wesley; 2001
- [75] Weng J. *Natural and Artificial Intelligence: Introduction to Computational Brain-Mind*. 2nd ed. Okemos, Michigan: BMI Press; 2019
- [76] Karpathy A, Toderici G, Shetty S, Leung T, Sukthankar R, Fei-Fei L. Large-scale video classification with convolutional neural networks. In: Proc. Computer Vision and Pattern Recognition, Columbus, Ohio, June 24–27, 2014. pp. 1-8
- [77] Weng J, Luciw M. Dually optimal neuronal layers: Lobe component analysis. *EEE Transactions on Autonomous Mental Development*. 2009;1(1):68-85
- [78] Weng J, Luciw MD. Brain-inspired concept networks: Learning concepts from cluttered scenes. *IEEE Intelligent Systems Magazine*. 2014; 29(6):14-22
- [79] Wood DJ, Bruner JS, Ross G. The role of tutoring in problem-solving. *Journal of Child Psychology and Psychiatry*. 1976:89-100
- [80] Burr S. Active learning literature survey. *Data Mining and Knowledge Discovery*. 1998;2(2):121-167
- [81] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*. 2017; 60(6):84-90
- [82] Weng J. Why deep learning's performance data are misleading. In: 2023 4th Int'l Conf. on Artificial Intelligence in Electronics Engineering, Haikou, China, Jan. 6–8 2023. NY: ACM Press. pp. 1-10. arXiv:2208.11228
- [83] Jain AK, Dubes RC. *Algorithms for Clustering Data*. New Jersey: Prentice-Hall; 1988
- [84] Wang Y, Wu X, Weng J. Synapse maintenance in the where-what network. In: Proc. Int'l Joint Conference on Neural Networks, San Jose, CA, July 31–August 5, 2011. NJ: IEEE Press. pp. 2823-2829
- [85] 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, Beijing, China, July 6–11 2014. NJ: IEEE Press. pp. 1-8
- [86] 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, Las Vegas, June 269 2016. NJ: IEEE Press. pp. 11-18
- [87] Gao Q, Ascoli GA, Zhao L. BEAN: Interpretable and efficient learning with biologically-enhanced artificial neuronal assembly regularization. *Frontiers in Neurobotics*. 2021;15:1-13. DOI: 10.3389/fnbot.2021.567482
- [88] Weng J. Life is science (36): Did Turing Awards go to fraud? Facebook blog [Accessed: March 8, 2020]. Available from: [www.facebook.com/juyang.weng/posts/10158319020739783](http://www.facebook.com/juyang.weng/posts/10158319020739783)
- [89] Weng J. Did Turing awards go to fraud? YouTube video [Accessed: June 4, 2020]. 1:04 hours. Available from: <https://youtu.be/Rz6CFIKrx2k>
- [90] Serre T, Wolf L, Bileschi S, Riesenhuber M, Poggio T. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2007;29(3):411-426
- [91] Bengio Y, LeCun Y, Hinton G. Deep learning for AI. *Communications of ACM*. 2021;64(7):58-65
- [92] Sermanet P, Kavukcuoglu K, Chintala S, LeCun Y. No more pesky learning rates. In: Proc. Int'l Conference on Machine Learning, Atlanta, GA, June 16–21 2013. pp. 343-351
- [93] Dauphin YN, Pascanu R, Gulcehre C, Cho K, Ganguli S, Bengio Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: *Advances in Neural Information Processing Systems*, Montreal, Canada. NY: Curran Associates, Inc.; 2014. pp. 2933-2941
- [94] Srivastava N, Hinton GE, Krizhevsky K, Sutskever I, Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*. 2014;15(1):1929-1958
- [95] Poggio T. Theoretical issues in deep networks. *Proceedings of the National Academy of Sciences*. 2020;117(48):30039-30045
- [96] Choromanska A, Henaff M, Mathieu M, Arous GB, LeCun Y. The loss surfaces of multilayer networks. In: Proc. *Machine Learning Research*. Vol. 38. NY: Curran Associates, Inc.; 2015. pp. 192-204
- [97] Kawaguchi K. Deep learning without poor local minima. Technical Report arXiv:1605.07110, MIT-CSAIL-TR-2016-005, Cambridge, MA [Accessed: May 23, 2016]
- [98] Weng J, Zheng Z, Wu X. Developmental Network Two, its optimality, and emergent Turing machines. U.S. Patent Application Number: 16265212 [Accessed: Feb. 1, 2019]. Approval pending
- [99] Knoll JA, Hoang VN, Honer J, Church S, Tran TH, Weng J. Optimal developmental learning for multisensory and multi-teaching modalities. In: Proc. *IEEE Int'l Conference on Development and Learning*, Pages 1–6, Beijing, China, Oct. 23–26, 2021
- [100] Weng J. A unified hierarchy for AI and natural intelligence through auto-programming for general purposes. *Journal of Cognitive Science*. 2020;21:53-102
- [101] Rabiner LR. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*. 1989;77(2):257-286

[102] Zheng Z, Wu X, Weng J. Emergent neural Turing machine and its visual navigation. *Neural Networks*. 2019;**110**: 116-130

[103] Wu X, Weng J. Muscle vectors as temporally “dense labels”. In: *Proc. Int’l Joint Conference on Neural Networks*, Glasgow, UK, July 19–24 2020. pp. 1-8

[104] Castro-Garcia J, Weng J. Emergent multilingual language acquisition using developmental networks. In: *Proc. Int’l Joint Conf. Neural Networks*, Budapest, Hungary, July 14–19 2019. NJ: IEEE Press. pp. 1-8

[105] Wu X, Weng J. On machine thinking. In: *Proc. Int’l Joint Conf. Neural Networks*, Shenzhen, China, July 18–22 2021. NJ: IEEE Press. pp. 1-8

IntechOpen