On Post Selection Using Test Sets (PSUTS) in AI

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

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

I. Introduction

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

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

A. Symbolic school

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

The symbolic school [2] assumes a micro-world in 4D space-time in which a set of objects or concepts, e.g., $S = \{o_1, o_2, ..., o_n\}$, is assumed to be uniquely defined among human programmers and their computers, represented by a series of symbols in time $\{o_1(t), o_2(t), ...o_n(t) \mid t_0 \le t < t_1\}$. The correspondences among all these symbols $\{o_i\}$ of the same object across different times are known as "the frame problem" [2] in AI which means that the programmer must manually link all such symbols together along time for each object. In computer vision, the symbolic school assumes a single symbol o_i, for all its 3D positions in its 3D trajectory $\{\mathbf{x}(t) \mid t_0 \le t \le t_1\}$ and uses techniques, such as feature tracking through video (e.g., for driverless cars). Therefore, the symbolic school is based on human-handcrafted set of symbols and their assumed meanings. Marvin Minsky wrote that symbols are "neat" [3], but in fact, symbols are "neat" mainly in human programmer's understanding but not in relating computer programs to a real world. A major problem that many AI researchers did not address is the generalization weakness of the symbolic school, as the following definition states:

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

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

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

J. Weng, "On Post Selections Using Test Sets (PSUTS) in AI," in Proc. International Joint Conference on Neural Networks, pp. 1-8, Shenzhen, China, July 18-22, 2021.

B. Connectionist School

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

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

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

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

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

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

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

C. Developmental School

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

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

$$f : X(t-1) \times Z(t-1) \mapsto Z(t), t = 1, 2, ... \tag{2}$$

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

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

II. PSUTS

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

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

A. Task-specific vs. Task-nonspecific

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

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

B. Batch vs. Incremental learning modes

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

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

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

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

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

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

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

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

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

C. Machine PSUTS

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

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

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

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

are trained each of which starts with a different set of random weights $\mathbf{w}_j$, using error backprop that locally and numerically minimizes the fitting error $f_{i,j}$ on the training set T. Grave et al. 2016 seems to have mentioned $n = 20$. Using the above example of $k = 10B$, $kn = 200$B a huge number that requires a lot of computational resources and manpower.

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

Definition 3 (Machine PSUVS): If the test set T' is not available, suppose the validation error of $N(\mathbf{a}_i, \mathbf{w}_j)$ is $e_{i,j}$ on the validation set V, find the best network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches the minimum validation error:

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

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

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

Definition 4 (Machine PSUTS): If the test set T' is available, suppose the test error of $N(\mathbf{a}_i, \mathbf{w}_j)$ is $e'_{i,j}$ on the test set T', find the best network $N(\mathbf{a}_{i^*}, \mathbf{w}_{j^*})$ so that it reaches the minimum test error:

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

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

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

Definition 5 (Distribution of errors of trained systems): The distributions of all kn trained networks' fitting errors $\{f_{ij}\}$, validation errors $\{e_{ij}\}$, and test errors $\{e'_{ij}\}$, $i = 1, 2, ...k$, $j = 1, 2, ...n$, as well as the values of k and n.

It is necessary to present some key statistical characteristics of such distributions. For example, ranked errors in decreasing order. Then given the maximum, 75%, 50%, 25%, and minimum value of these kn values for the fitting errors, validation errors. and test errors, so that the research community can see whether error-backprop can avoid local minima in deep learning. For transparency, it seems necessary to report such distribution characteristics other than the minimum value e'_{i^*,j^*}.

They should report such distribution characteristics to reflect the effects of machine PSUTS. Our experience in experiments indicated that the maximum and the minimum values of the distribution of fitting errors alone are drastically different, around 80% and 5%, respectively. Section III will discuss why.

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

The architecture parameter vector $\mathbf{a}_{i^*}$ and weights $\mathbf{w}_{j^*}$ overfits T, V and T'. If an unobserved data set T'', disjoint with T', $T' \cap T'' = \emptyset$, is observed from a new environment, the error rate e''_{i^*,j^*} of $N(\mathbf{a}_{i*}, \mathbf{w}_{j^*})$ is predicted to significantly higher than e'_{i^*,j^*},

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

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

D. Human PSUTS

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

Definition 6 (Human PSUTS): After planning or knowing what will be in the training set T and test set T', a human post-selects features in networks instead of using a machine to learn such features.

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

Unfortunately, almost all methods in the symbolic school use Human PSUTS because it is always the same human who plans for and design a micro-world and collect the test set T'. The key to an acceptable test score lies in how much detail the human designer can plan for what is in the test sets.

III. Why Error Backprop Needs PSUTS

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

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

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

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

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

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

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

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

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + w_2 \mathbf{y} \tag{7}$$

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

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

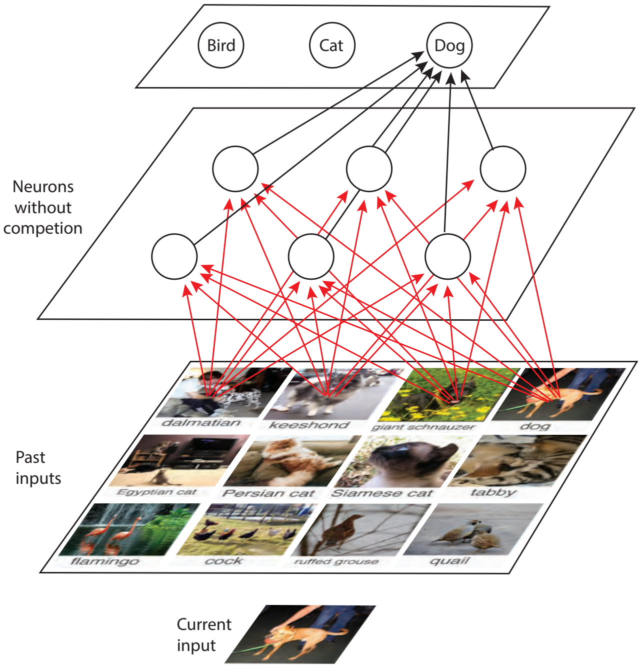

Fig. 1. Lack of role-determination in hidden neurons.

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

The meaning (3) of Theorem 2 are illustrated by Fig. 1. CNNs do not have a competition mechanism in hidden layers. Complete connections initialized with random weights are provided for all consecutive areas (also called layers), from input area all the way to the output area. If the z_j neuron is in the output motor area and each output neuron is assigned a single class label, the role of z_j ("dog" in the figure) is determined by human supervised label "dog". However, let us assume instead that z_j is in a hidden area, not responsible for the "dog" class. z_j still updates its input weights using the gradient. Likewise, the pre-synaptic area Y, is labeled "neurons without competition". The hidden neurons in this area do not have a competition mechanism to decide the role of each neuron there. This analysis leads us to the following theorem.

Theorem 3 (Random Roles in Error-BackProp): A set of random initial weights in a network assigns random roles to all hidden neurons, from which a local minimal point based on error-backprop learning inherits this particular random-role assignment. Which neurons in each hidden areas take a role does not matter, but how hidden neurons share a set of roles in each hidden area does matter.

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

Theorem 4 (Percentage Luck of Error-BackProp): Suppose a CNN has $l > 1$ areas, $A_0, A_1, ..., A_l$, connected by a cascade or a variation thereof. A_0 takes input frames $\{\mathbf{x} \in X\}$ and A_l is the output area for classification. It has a total of m hidden neurons in an area that share a common receptive field R in A_0. Let the percentage of the m hidden neurons do not fire given an input frame $\mathbf{x}$ be denoted as $p(\mathbf{x})$. Then, the Error-BackProp depends on the average $\bar{p} = E_{\mathbf{x}\in X}\{p(\mathbf{x})\}$ to be a reasonably small value, called the percentage luck.

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

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

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

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

IV. Developmental Errors

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

A. Developmental Errors

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

Definition 7 (Developmental error): The developmental errors of a developmental network $N = (X, Y, Z, M)$ with sensory area X, skull closed hidden area Y and motor area Z and memory M, runs through lifetime by sampling at discrete time indices $t = 0, 1, 2, ...$ as $N(t) = (X(t), Y(t), Z(t), M(t))$. Start at inception $t = 0$ with supervised sensory input $\mathbf{x}_0 \in X(0)$, initial state $\mathbf{z}_0 \in Z(0)$, and randomly initialized weigh vector $\mathbf{y}_0 \in Y(0)$, and initial memory $\mathbf{m}_0 \in M(0)$. At each time $t = 1, 2, ...$, the network N recursively and incrementally updates:

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

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

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

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

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

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

may be significantly different from the nearest matched learned weight vectors of all hidden neurons. If that happens and there are still new hidden neuron that have not fired, a new hidden neuron is spawned that perfectly memorize this new context regardless its randomly initialized weights. When all the available hidden neurons have fired at least once, the DN will update the top-k matched neurons optimally in the sense of maximum likelihood (ML), as proven in [13]. For more specific time periods, such as the period from time t_1 to t_2 during which only disjoint tests were made by the teacher and the learning agent is not motor-supervised, the average error is denoted as $\bar{e}(t_1 : t_2)$. Therefore, $\bar{e}(t)$ means $\bar{e}(0 : t)$.

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

B. Competition

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

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

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

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

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

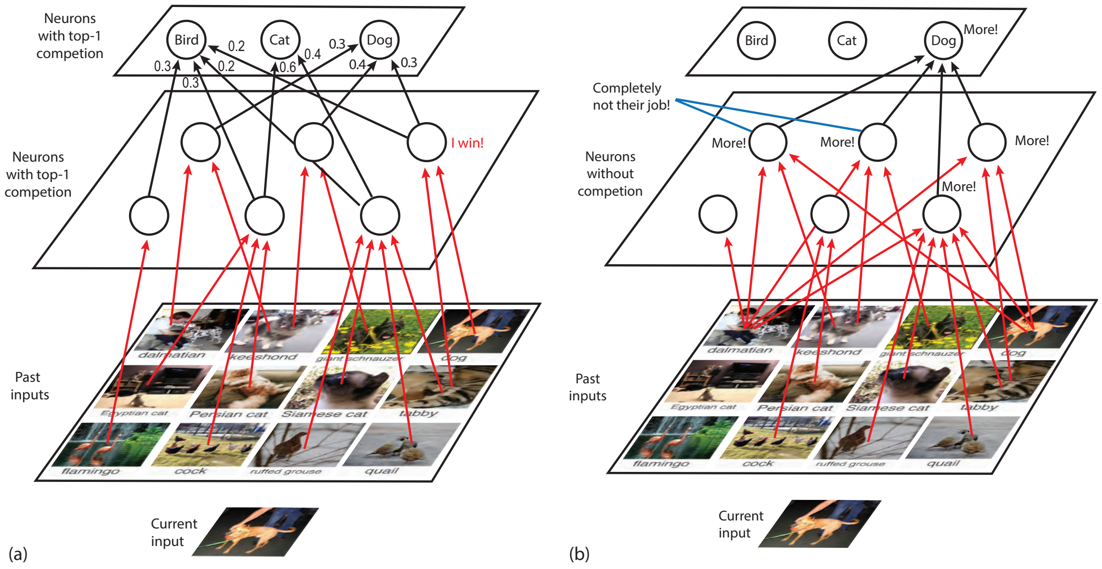

Fig. 2. How competition automatically assigns roles among hidden neurons without a central controller: The case for automatically construct a mapping $f : X \mapsto S$. (a) Each hidden neuron must win-and-fire for multiple inputs. (b) Error-backprop from the "dog" motor neuron asks some hidden neurons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

clustering based on top-k competition. The weight vector of each hidden Y neuron corresponds to a cluster in the X space. In Fig. 2(a), $k = 1$ for top-k competition in Y.

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

The Candid Covariance-free Incremental (CCI) Lobe Component Analysis (LCA) in Weng 2009 [28] proved that such automatic assignment of roles through competition results in a dually optimal neuronal layer, optimal spatially and optimal temporally. Optimal spatially means the CCI LCA incrementally computes the first principal component features of the receptive field. Optimal temporally means that the principal component vector has the least expected distance to its target—the optimal estimator in the sense of minimum variance to the true principal component vector.

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

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

It has been proven that the above expression incrementally computes the first principal component as $\mathbf{v}$. The learning rate $w_2 = \frac{1}{a}$ is the optimal and age-dependent learning rate. CCI LCA is a framework for dually optimal Hebbian learning. The property "candid" corresponds to the property that sum of the learning rate $w_2 = \frac{1}{a}$ and the retention rate $w_1 = 1 - \frac{1}{a}$ is always 1 to keep the "energy" of response r weighted input $\mathbf{x}$ unchanged (e.g., not to explode or vanish). This dually optimality resolves the three problems in Theorem 2.

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

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

2) Sensorimotor networks: The main reason that Marvin Minsky [3] complained that neural network is scruffy was because conventional neural networks lacked not only the optimality described above for sensory networks, but also lacked the Emergent Universal Turing Machines (EUTM) that is ML-optimal [13].

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

C. FA as sensorimotor mapping

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

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

Weng [13] extended the definition an FA so that it outputs its state so the resulting FA becomes an Agent FA (AFA). Further, Weng [13] extended the action q to the machinery of Turing machine so that action q includes output symbol to the Turing tape and the head motion of the read-write head of a Turing machine. With this extension, Weng [13] proved that any Turing machine is an AFA.

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

V. Conclusions

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

References

[1] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433–460, October 1950.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice-Hall, 3rd ed., 2010.

[3] M. Minsky, "Logical versus analogical or symbolic versus connectionist or neat versus scruffy," *AI Magazine*, vol. 12, no. 2, pp. 34–51, 1991.

[4] J. Weng, T. S. Huang, and N. Ahuja, *Motion and Structure from Image Sequences*. New York: Springer-Verlag, 1993.

[5] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Systems, Man and Cybernetics*, vol. 13, no. 5, pp. 826–834, 1983.

[6] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Chichester: Wiley, 1994.

[7] T. Serre, T. Poggio, M. Riesenhuber, L. Wolf, and S. Bileschi, "High-performance vision system exploiting key features of visual cortex," *US Patent*, vol. US7606777B2, Sept. 1 2006.

[8] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

[11] Y. LeCun, L. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[12] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen, "Autonomous mental development by robots and animals," *Science*, vol. 291, no. 5504, pp. 599–600, 2001.

[13] J. Weng, "Brain as an emergent finite automaton: A theory and three theorems," *International Journal of Intelligent Science*, vol. 5, no. 2, pp. 112–131, 2015.

[14] J. Weng, N. Ahuja, and T. S. Huang, "Learning recognition and segmentation using the Cresceptron," *International Journal of Computer Vision*, vol. 25, pp. 109–143, Nov. 1997.

[15] J. Weng, "Why have we passed "neural networks do not abstract well"?," *Natural Intelligence: the INNS Magazine*, vol. 1, no. 1, pp. 13–22, 2011.

[16] C. M. Super, "Environmental effects on motor development: A case of Africa infant precocity," *Developmental Medicine and Child Neurology*, vol. 18, pp. 561–567, 1976.

[17] K. A. Thoroughman and J. A. Taylor, "Rapid reshaping of human motor generalization," *Jounal of Neuroscience*, vol. 25, no. 39, pp. 8948–8953, 2005.

[18] G. Rizzotti, L. Riggio, I. Dascola, and C. Umilta, "Reorienting attention across the horizontal and vertical meridians: evidence in favor of a premotor theory of attention," *Neuropsychologia*, vol. 25, pp. 31–40, 1987.

[19] T. Moore, K. M. Armstrong, and M. Fallah, "Visuomotor origins of covert spatial attention," *Neuron*, vol. 40, pp. 671–683, 2003.

[20] J. M. Iverson, "Developing language in a developing body: the relationship between motor development and language development," *Journal of child language*, vol. 37, no. 2, pp. 229–261, 2010.

[21] J. Weng and M. Luciw, "Brain-like emergent spatial processing," *IEEE Trans. Autonomous Mental Development*, vol. 4, no. 2, pp. 161–185, 2012.

[22] J. Weng, M. Luciw, and Q. Zhang, "Brain-like temporal processing: Emergent open states," *IEEE Trans. Autonomous Mental Development*, vol. 5, no. 2, pp. 89 – 116, 2013.

[23] J. Weng, Z. Zheng, X. Wu, and J. Castro-Garcia, "Auto-programming for general purposes: Theory and experiments," in *Proc. International Joint Conference on Neural Networks*, (Glasgow, UK), pp. 1–8, July 19-24 2020.

[24] J. Weng, "Autonomous programming for general purposes: Theory," *International Journal of Huamnoid Robotics*, vol. 17, pp. 1–36, August 2020.

[25] J. Weng, "Conscious intelligence requires developmental autonomous programming for general purposes," in *Proc. IEEE International Conference on Development and Learning and Epigenetic Robotics*, (Valparaiso, Chile), pp. 1–7, Oct. 26-27 2020.

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

[27] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. Computer Vision and Pattern Recognition*, (Columbus, Ohio), pp. +1–8, June 24-27, 2014.

[28] J. Weng and M. Luciw, "Dually optimal neuronal layers: Lobe component analysis," *IEEE Trans. Autonomous Mental Development*, vol. 1, no. 1, pp. 68–85, 2009.

[29] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411–426, 2007.

[30] Z. Zheng and J. Weng, "Mobile device based outdoor navigation with on-line learning neural network: a comparison with convolutional neural network," in *Proc. 7th Workshop on Computer Vision in Vehicle Technology (CVVT 2016) at CVPR 2016*, (Las Vega), pp. 11–18, June 269 2016.

[31] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, "No more pesky learning rates," in *Proc. International Conference on Machine learning*, (Atlanta, GA), pp. 343–351, June 16-21 2013.

[32] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems*, (Montreal, Canada), pp. 2933–2941, Curran Associates, Inc., 2014.

[33] N. Srivastava, G. E. Hinton, K. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overtting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[34] T. Poggio, "Theoretical issues in deep networks," *Proceedings of thr National Academy of Sciences*, vol. 117, no. 48, pp. 30039–30045, 2020.