NANYANG
TECHNOLOGICAL
UNIVERSITY

Extreme Learning Machines (ELM)

Outline

- **Part I - ELM Philosophy and Generalized SLFN Cases:**
  - Neural networks and machine learning history
  - Rethink machine learning and artificial intelligence
  - Philosophy and belief of Extreme Learning Machines (ELM)
    - Do we really need so many different type of learning algorithms for so many type of networks (various types of SLFNs, regular and irregular multi-layers of networks, various type of neurons)?
    - Can the gap between machine learning and biological learning be filled?
    - Should learning be transparent or of blackbox?
    - SVM provides suboptimal solutions.
  - Machine learning and Internet of Things
  - Machine intelligence and human intelligence

Outline

- **Part II – Hierarchical ELM**
  - Unsupervised/semi supervised ELM
  - Feature learning
  - Hierarchical ELM
  - ELM + (other algorithms)
- **Part III – ELM Theories and Open Problems**
  - ELM theories:
    - Universal approximation capability
    - Classification capability
  - Incremental learning
  - Online sequential learning
  - Open problems

NANYANG TECHNOLOGICAL UNIVERSITY

Part I

ELM Philosophy and Generalized SLFN Cases

Frank Rosenblatt: Perceptron

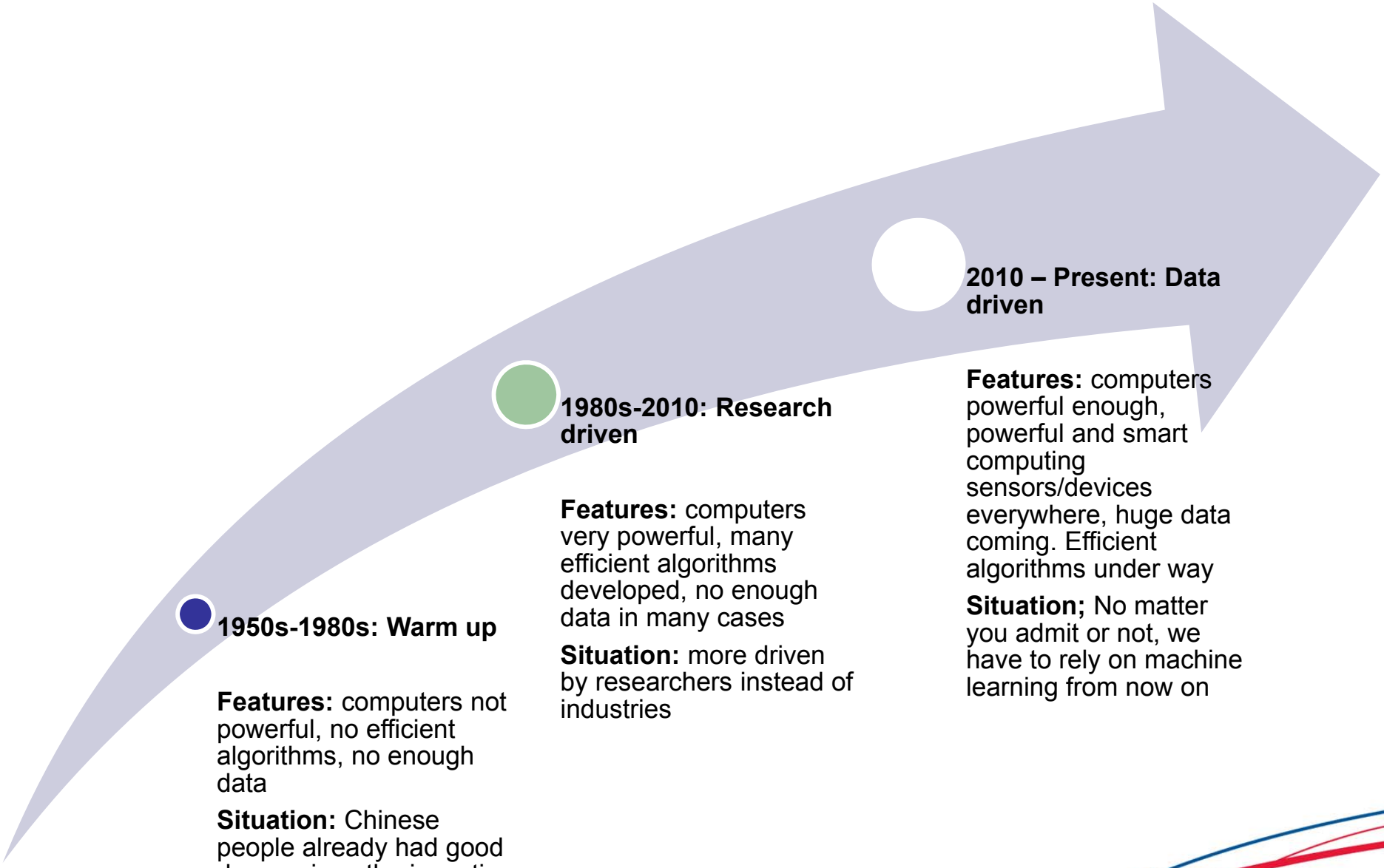
- **Cognition Dream in 60 Years Ago …**
  - "Rosenblatt made statements about the perceptron that caused a heated controversy among the fledgling AI community."
  - *Cognition*: "Based on Rosenblatt's statements, The New York Times reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to **walk**, **talk**, **see**, **write**, **reproduce** itself and be **conscious** of its existence"

http://en.wikipedia.org/wiki/Perceptron

Perceptron and AI Winter

- **"AI Winter" in 1970s**
  - "Beautiful mistakes" [Minsky and Papert 1969]: Minsky claimed in his book that the simple XOR cannot be resolved by two-layer of feedforward neural networks, which "drove research away from neural networks in the 1970s, and contributed to the so-called AI winter." [Wikipedia2013]

Three Waves of Machine Learning



1950s-1980s: Warm up

Features: computers not powerful, no efficient algorithms, no enough data

Situation: Chinese people already had good dream since the inception of computers and called computers as “Electronic Brains （电脑）”

1980s-2010: Research driven

Features: computers very powerful, many efficient algorithms developed, no enough data in many cases

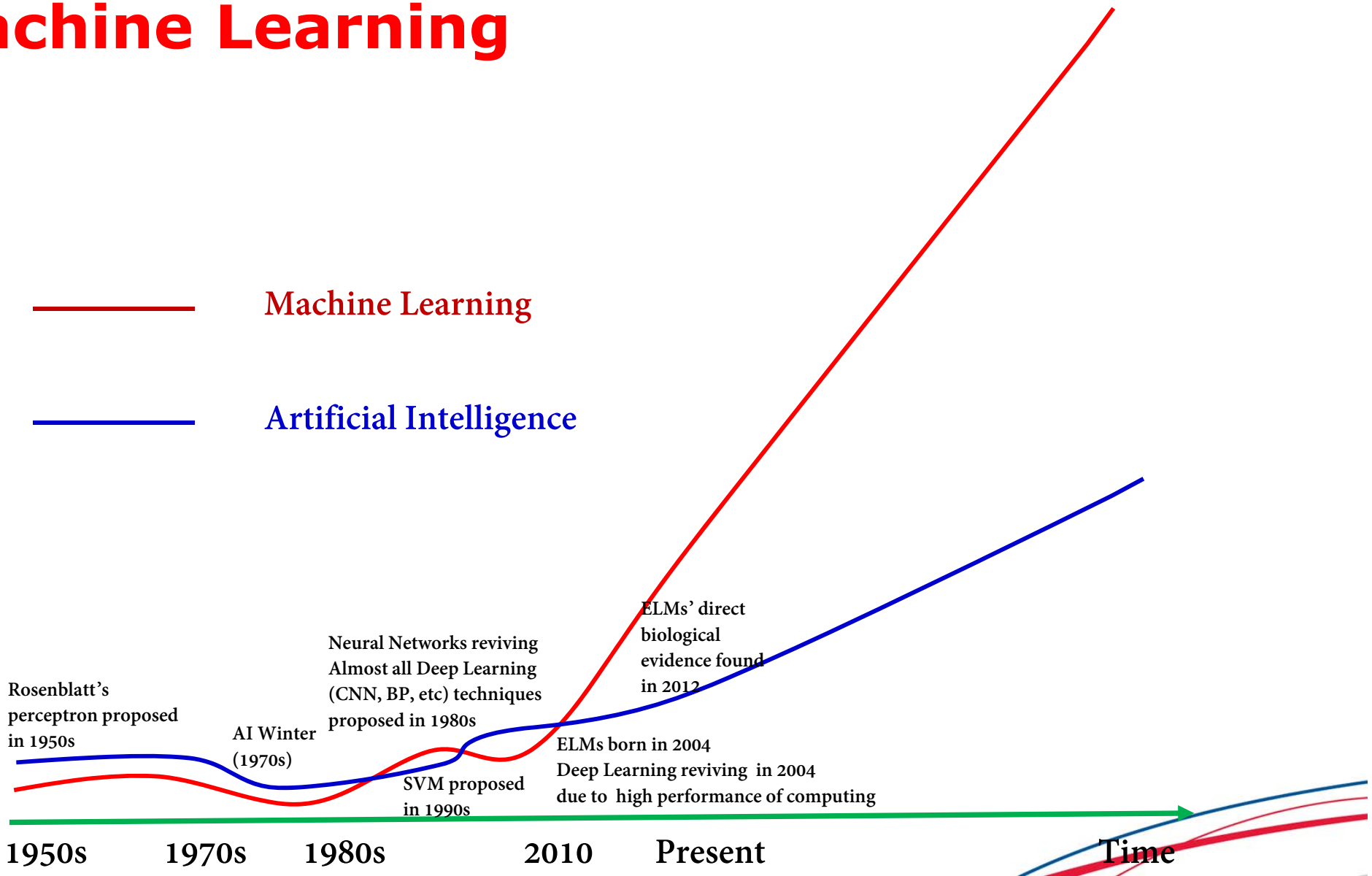
Situation: more driven by researchers instead of industries

2010 – Present: Data driven

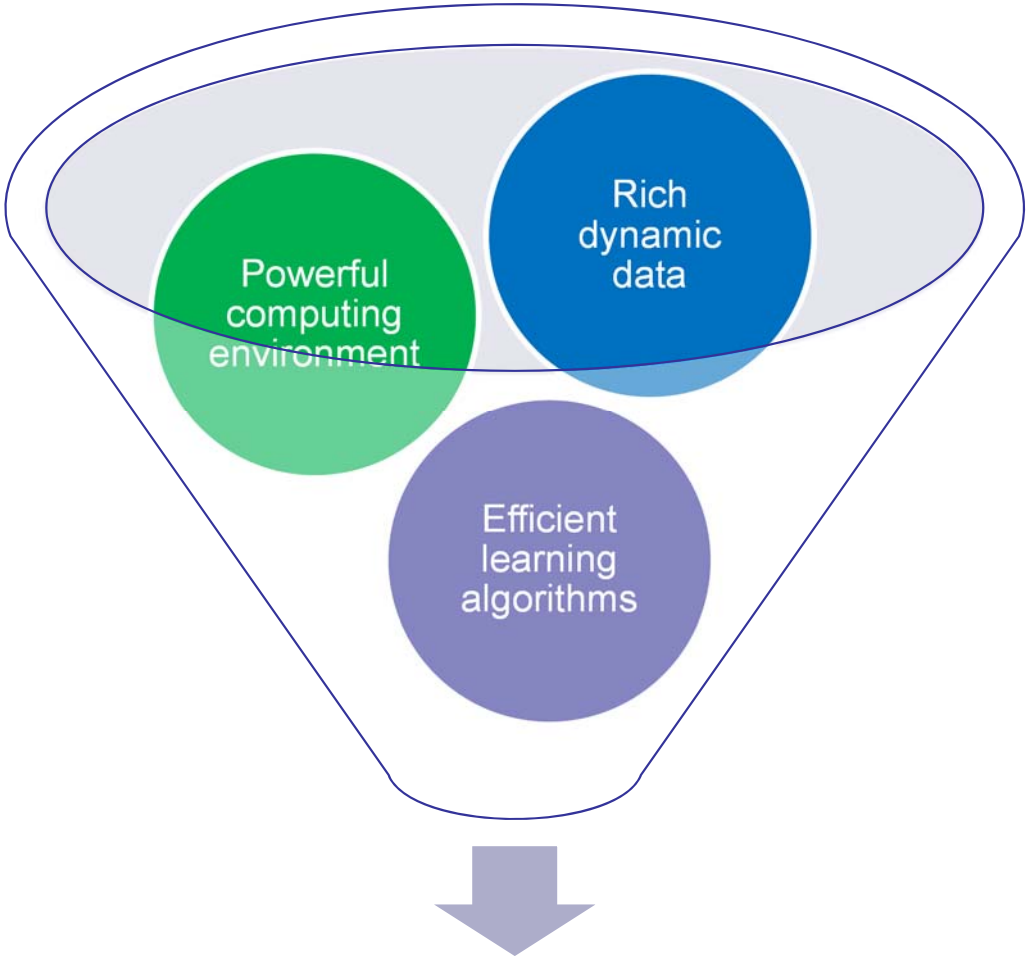
Features: computers powerful enough, powerful and smart computing sensors/devices everywhere, huge data coming. Efficient algorithms under way

Situation; No matter you admit or not, we have to rely on machine learning from now on

Rethink Artificial Intelligence and Machine Learning

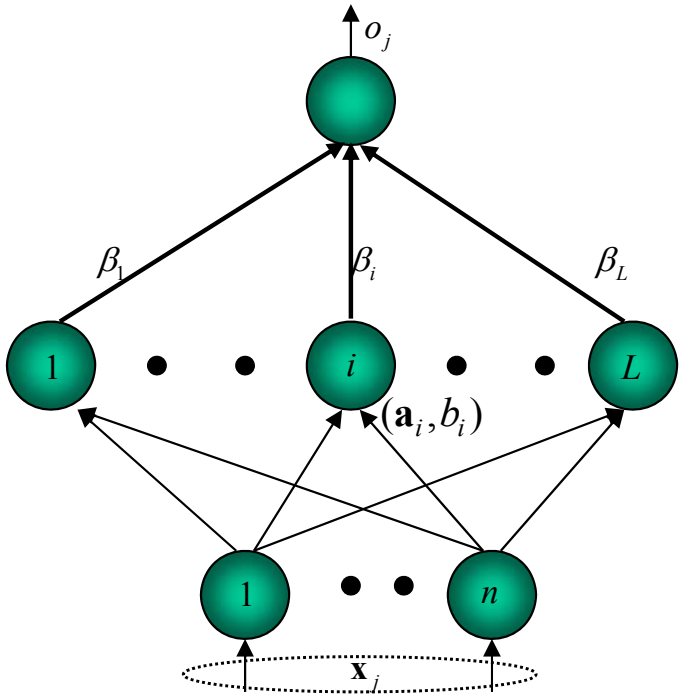


Necessary Conditions of Machine Learning Era



Three necessary conditions of true machine learning era, which have been fulfilling since 2010

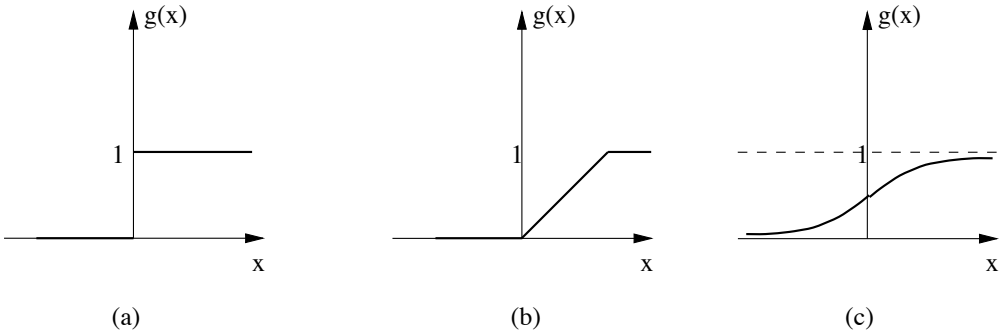
Feedforward Neural Networks



Output Node

L Hidden Nodes

n Input Nodes



Output of additive hidden nodes:

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i)$$

Output of RBF hidden nodes:

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|)$$

The output function of SLFNs is:

$$f_L(x) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\mathbf{a}_i, b_i, \mathbf{x})$$

$\boldsymbol{\beta}_i$: Output weight vector connecting the ith hidden node and the output nodes

Feedforward Neural Networks

- **Mathematical Model**
  - Approximation capability [Leshno 1993, Park and Sandberg 1991]: Any continuous target function $f(\mathbf{x})$ can be approximated by SLFNs with **adjustable hidden nodes**. In other words, given any small positive value ε, for SLFNs with enough number of hidden nodes (L) we have $\|f_L(\mathbf{x}) - f(\mathbf{x})\| < \varepsilon$.
  - Classification capability [Huang, et al 2000]: As long as SLFNs can approximate any continuous target function $f(\mathbf{x})$, such SLFNs can differentiate any disjoint regions.

M. Leshno, et al., "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861-867, 1993.
J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, pp. 246-257, 1991.
G.-B. Huang, et al, "Classification ability of single hidden layer feedforward neural networks," *IEEE Trans. Neural Networks*, vol. 11, no. 3, pp. 799–801, May 2000.

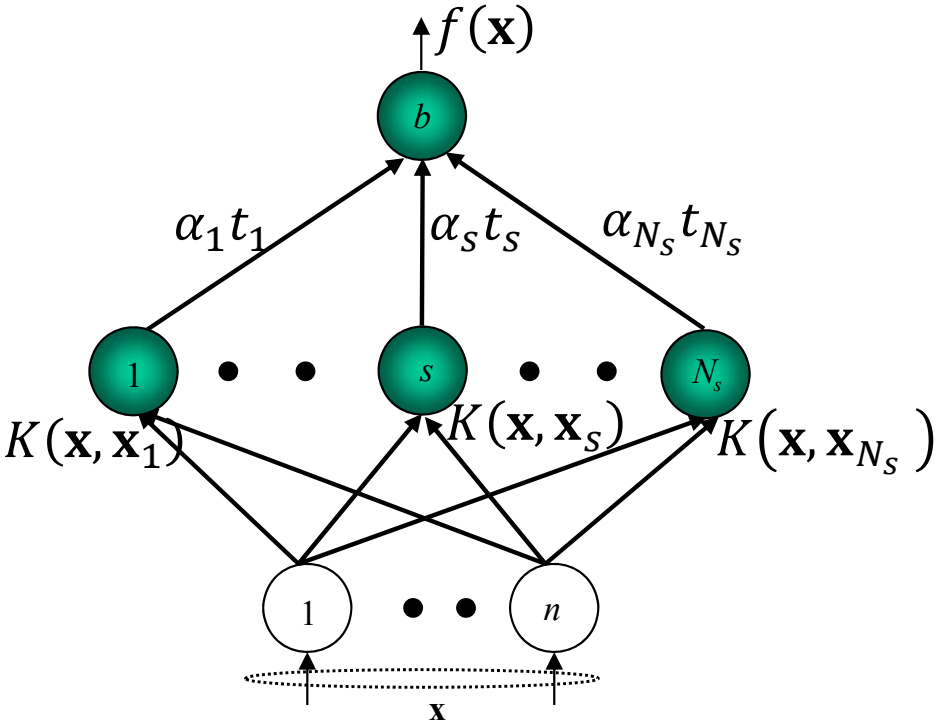
Feedforward Neural Networks

- **Learning Issue**
  - Conventional theories: only resolves the **existence** issue, however, does not tackle learning issue at all.
  - In real applications, target function f is usually unknown. One wishes that unknown f could be approximated by SLFNs f_L appropriately.

Feedforward Neural Networks

- **Learning Methods**
  - Many learning methods mainly based on gradient-descent / iterative approaches have been developed over the past three decades.
    - Back-Propagation (BP) [Rumelhart 1986] and its variants are most popular.
  - Least-square (LS) solution for RBF network, with single impact factor for all hidden nodes. [Broomhead and Lowe 1988]
  - QuickNet (White, 1988) and Random vector functional network (RVFL) [Igelnik and Pao 1995]
  - Support vector machines and its variants. [Cortes and Vapnik 1995]
  - Deep learning: dated back to 1960s and resurgence in mid of 2000s [wiki 2015]

Support Vector Machine – an Alternative Solution of SLFN



SVM optimization formula

$$\text{minimize: } L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{subject to: } t_i(\mathbf{w}\cdot\phi(\mathbf{x}) + b) \geq 1 - \xi_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

LS-SVM optimization formula

$$\text{minimize: } L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\xi_i^2$$

$$\text{subject to: } t_i(\mathbf{w}\cdot\phi(\mathbf{x}) + b) = 1 - \xi_i, \forall i$$

The decision function of SVM and LS-SVM is:

$$f(x) = \text{sign}\left(\sum_{s=1}^{N_s}\alpha_s t_s K(\mathbf{x}, \mathbf{x}_s) + b\right)$$

Typical kernel function:

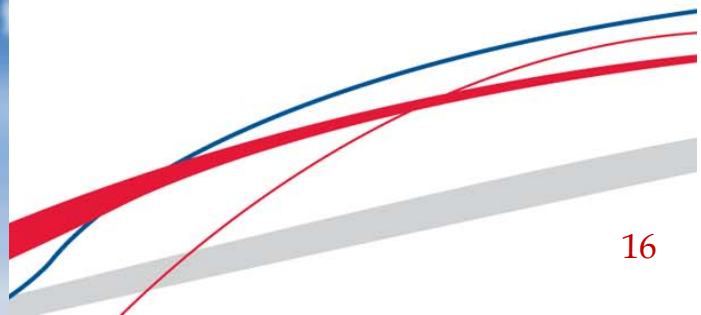
$$K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma\|\mathbf{x} - \mathbf{y}\|^2)$$

Feedforward Neural Networks

- **Interesting 20 Years of Cycles**
  - Robenblatt's Perceptron proposed in mid of 1950s, sent to "Winter" in 1970s
  - Back-Propagation (BP) proposed in 1970s, reaching research peak in mid of 1990s
  - Support vector machines proposed in 1995, reaching research peak early this century.
- **There are exceptional cases:**
  - E.g, most deep learning algorithms proposed in 1960s ~1980s, becoming popular only since 2010 (more or less)

Research in Neural Networks Stuck …?

Conventional Learning Methods	Biological Learning
Very sensitive to network size	Stable in a wide range (tens to thousands of neurons in each module)
Difficult for parallel implementation	Parallel implementation
Difficult for hardware implementation	"Biological" implementation
Very sensitive to user specified parameters	Free of user specified parameters
Different network types for different type of applications	One module possibly for several types of applications
Time consuming in each learning point	Fast in micro learning point
Difficult for online sequential learning	Nature in online sequential learning
"Greedy" in best accuracy	Fast speed and high accuracy
"Brains (devised by conventional learning methods)" are chosen after applications are present	Brains are built before applications



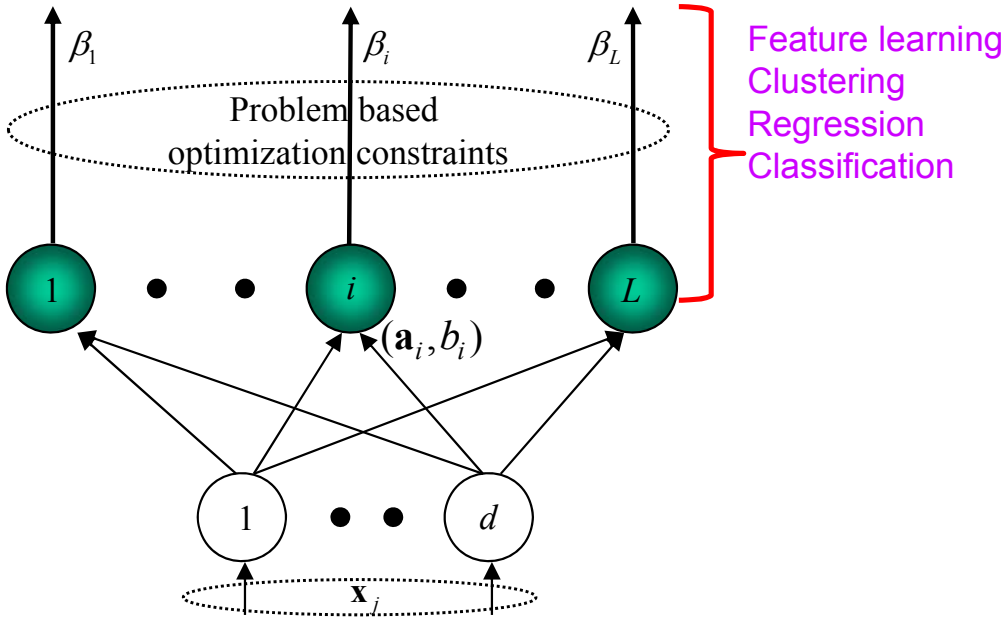
Research in Neural Networks Stuck …?

- **Reasons**
  - Based on the conventional existence theories:
    - Since hidden nodes are important and critical, we need to find some way to **adjust hidden nodes**.
    - Learning focuses on hidden nodes.
    - Learning is tremendously inefficient.
  - Intensive research: many departments/groups in almost every university/research institution have been spending huge manpower on looking for so-called "appropriate" (actually still very basic) learning methods in the past 30 years.
- **Question**
  - Is free lunch really impossible?
  - The answer is "seemingly far away, actually close at hand and right under nose" "远在天边, 近在眼前"

Fundamental Problems to Be Resolved by Extreme Learning Machines (ELM)

- **Do we really need so many different types of learning algorithms for so many different types of networks?**
  - different types of SLFNs
    - sigmoid networks
    - RBF networks
    - polynomial networks
    - complex (domain) networks
    - Fourier series
    - wavelet networks, etc
  - multi-layers of architectures
- **Do we really need to tune wide type of hidden neurons including biological neurons (even whose modeling is unknown) in learning?**

Extreme Learning Machines (ELM)



L Random Hidden Neurons (which need not be algebraic sum based) or other ELM feature mappings. Almost any nonlinear piecewise continuous hidden nodes:

$h_i(\mathbf{x}) = G_i(\mathbf{a}_i, b_i, \mathbf{x})$

Although we don't know biological neurons' true output functions, most of them are nonlinear piecewise continuous, covered by ELM theories.

Output function of "generalized" SLFNs:

$$f_L(x) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x})$$

The hidden layer output function (hidden layer mapping, ELM feature space):

$$\boldsymbol{h}(x) = [G(\mathbf{a}_1, b_1, \mathbf{x}), \cdots, G(\mathbf{a}_L, b_L, \boldsymbol{x})]$$

The output functions of hidden nodes can be but are not limited to:

Sigmoid: $G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i)$

RBF: $G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|)$

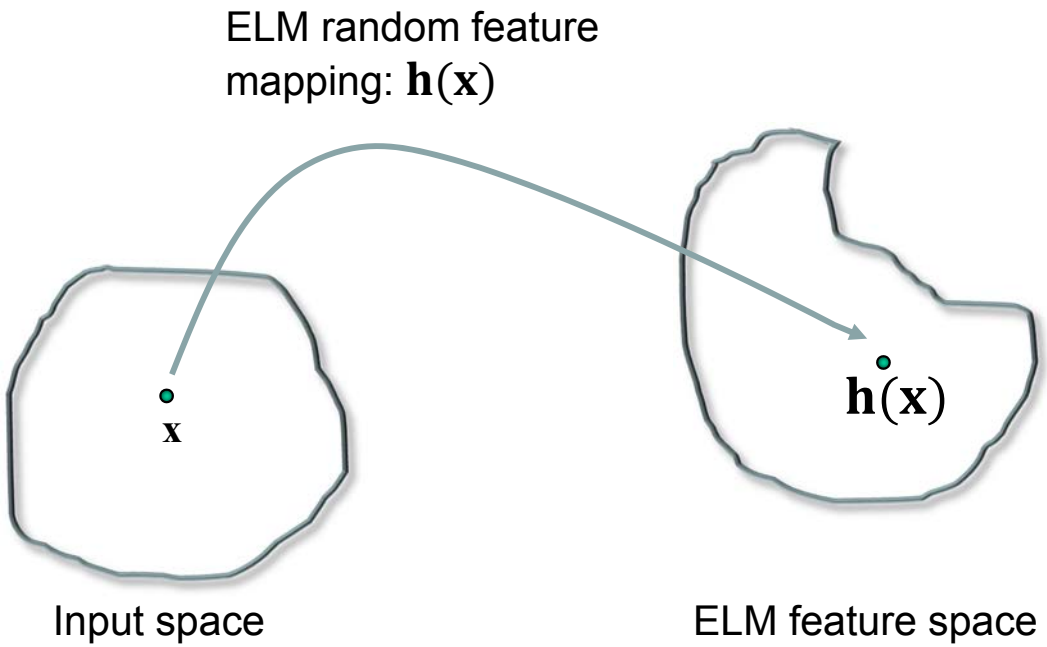
Fourier Series: $G(\mathbf{a}_i, b_i, \mathbf{x}) = \cos(\mathbf{a}_i \cdot \mathbf{x} + b_i)$

Extreme Learning Machines (ELM)

- **New Learning Theory** - *Learning Without Iteratively Tuning Hidden Neurons in general architectures*: Given any nonconstant piecewise continuous function g, if continuous target function $f(\mathbf{x})$ can be approximated by SLFNs with adjustable hidden nodes g then the hidden node parameters of such SLFNs needn't be tuned. [Huang, et al 2006, 2007]
  - It not only proves the **existence** of the networks but also provides **learning solutions**.
  - All these hidden node parameters can be randomly generated without training data.
  - That is, for any continuous target function $f(\mathbf{x})$ and any randomly generated sequence $\{(\mathbf{a}_i, b_i)_{i=1}^{L}\}$, $\lim_{L\to\infty} \|f(\mathbf{x}) - f_L(\mathrm{x})\| = \lim_{L\to\infty} \left\|f(\mathbf{x}) - \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\mathbf{a}_i, b_i, \mathbf{x})\right\| = 0$ holds with probability one if $\boldsymbol{\beta}_i$ is chosen to minimize $\|f(\mathbf{x}) - f_L(\mathbf{x})\|$, $\forall i$. [Huang, et al 2006]
- Direct biological evidence later found in 2013 [Fusi, 2013]

G.-B. Huang, et al., "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879-892, 2006.
G.-B. Huang and L. Chen, "Convex Incremental Extreme Learning Machine," *Neurocomputing*, vol. 70, pp. 3056-3062, 2007.
O. Barak, et al, "The importance of mixed selectivity in complex cognitive tasks," *Nature*, vol.497, pp. 585-590, 2013
M. Rigotti, et al, "The sparseness of mixed selectivity neurons controls the generalization-discrimination trade-off," *Journal of Neuroscience*, vol. 33, no. 9, pp. 3844-3856, 2013

Extreme Learning Machines (ELM)



The hidden layer output function (hidden layer mapping, ELM feature space):

$$\boldsymbol{h}(x) = [G(\mathbf{a}_1, b_1, \mathbf{x}), \cdots, G(\mathbf{a}_L, b_L, \boldsymbol{x})]$$

The output functions of hidden nodes can be but are not limited to

Sigmoid: $G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i)$

RBF: $G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|)$

Fourier Series: $G(\mathbf{a}_i, b_i, \mathbf{x}) = \cos(\mathbf{a}_i \cdot \mathbf{x} + b_i)$

L Random Hidden Neurons (which need not be algebraic sum based) or other ELM feature mappings. Almost any nonlinear piecewise continuous hidden nodes:

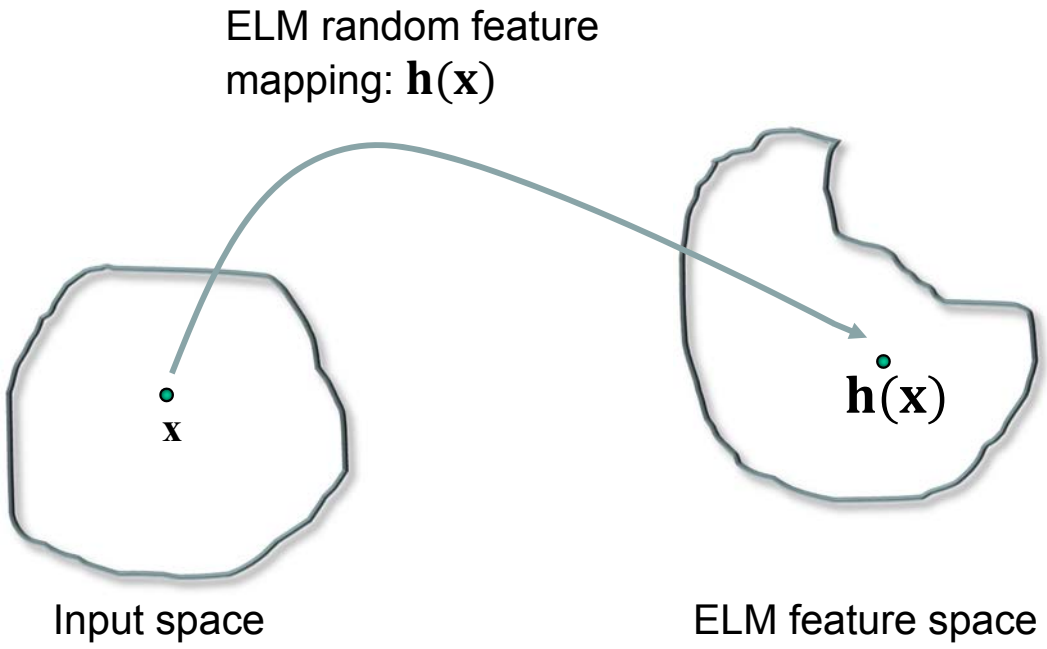
$h_i(\mathbf{x}) = G_i(\mathbf{a}_i, b_i, \mathbf{x})$

Although we don't know biological neurons' true output functions, most of them are nonlinear piecewise continuous, covered by ELM theories.

Conventional Random Projection is just a specific case of ELM random feature mapping (ELM feature space) when linear additive hidden node is used.

Random Projection: $G(\mathbf{a}_i, b_i, \mathbf{x}) = \mathbf{a}_i \cdot \mathbf{x}$

Extreme Learning Machines (ELM)



The hidden layer output function (hidden layer mapping, ELM feature space):

$$\boldsymbol{h}(x) = [G(\mathbf{a}_1, b_1, \mathbf{x}), \cdots, G(\mathbf{a}_L, b_L, \boldsymbol{x})]$$

The output functions of hidden nodes can be but are not limited to

Sigmoid: $G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i)$

RBF: $G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|)$

Fourier Series: $G(\mathbf{a}_i, b_i, \mathbf{x}) = \cos(\mathbf{a}_i \cdot \mathbf{x} + b_i)$

Convolutional nodes

Almost any nonlinear piecewise continuous hidden nodes: $h_i(\mathbf{x}) = G_i(\mathbf{a}_i, b_i, \mathbf{x})$, including sigmoid networks, RBF networks, trigonometric, networks, threshold networks, fuzzy inference systems, fully complex, neural networks, high-order networks, ridge polynomial networks, wavelet networks, *convolutional neural networks*, etc..

Conventional Random Projection is just a specific case of ELM random feature mapping (ELM feature space) when linear additive hidden node is used.

Random Projection: $G(\mathbf{a}_i, b_i, \mathbf{x}) = \mathbf{a}_i \cdot \mathbf{x}$

Extreme Learning Machines (ELM)

- **Essence of ELM**
  - Hidden layer need not be tuned.
    - "randomness" is just one of ELM's implementation, but not all
    - Some conventional methods adopted "semi-randomness"
  - Hidden layer mapping $\boldsymbol{h}(\mathbf{x})$ satisfies universal approximation conditions.
  - Minimize:$\|\mathbf{H}\boldsymbol{\beta}-\mathbf{T}\|_p$ and $\|\boldsymbol{\beta}\|_q$
    - (norm p and q could have different values, $q=1,\frac{1}{2},2,\cdots$)
  - It satisfies both ridge regress theory [Hoerl and Kennard 1970] and neural network generalization theory [Bartlett 1998].
  - It fills the gap and builds bridge among neural networks, SVM, random projection, Fourier series, matrix theories, linear systems, etc.

Basic ELM – a L2 Norm Solution

- **Three-Step Learning Model** [Huang, et al 2004, 2006]

Given a training set $\{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, and the number of hidden nodes L,

1) Assign randomly hidden node parameters$(\mathbf{a}_i, b_i), i = 1, \cdots, L$.

2) Calculate the hidden layer output matrix $\mathbf{H} = \begin{bmatrix} \boldsymbol{h}(\mathbf{x}_1) \\ \vdots \\ \boldsymbol{h}(\mathbf{x}_N) \end{bmatrix}$.

3) Calculate the output weights $\boldsymbol{\beta}$.

ELM Web portal: www.extreme-learning-machines.org

Extreme Learning Machines (ELM)

- **Salient Features**
  - "Simple Math is Enough." ELM is a simple tuning-free three-step algorithm.
  - The learning speed of ELM is extremely fast.

  - Unlike conventional existence theories, *the hidden node parameters are not only independent of the training data but also of each other*. Although hidden nodes are important and critical, they need not be tuned.
  - Unlike conventional learning methods which MUST see the training data before generating the hidden node parameters, *ELM could generate the hidden node parameters before seeing the training data*.
  - Homogenous architectures for compression, feature learning, clustering, regression and classification.

Extreme Learning Machines (ELM)

- **Ridge regression theory based ELM**

$$\boldsymbol{f}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\boldsymbol{\beta} = \boldsymbol{h}(\mathbf{x})\mathbf{H}^{\boldsymbol{T}}\left(\mathbf{H}\mathbf{H}^{\boldsymbol{T}}\right)^{-\mathbf{1}}\mathbf{T} \Rightarrow \boldsymbol{h}(\mathbf{x})\mathbf{H}^{\boldsymbol{T}}\left(\frac{I}{C} + \mathbf{H}\mathbf{H}^{\boldsymbol{T}}\right)^{-\mathbf{1}}\mathbf{T}$$

and

$$\boldsymbol{f}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\boldsymbol{\beta} = \boldsymbol{h}(\mathbf{x})\left(\mathbf{H}^{\boldsymbol{T}}\mathbf{H}\right)^{-\mathbf{1}}\mathbf{H}^{\boldsymbol{T}}\mathbf{T} \Rightarrow \boldsymbol{h}(\mathbf{x})\left(\frac{I}{C} + \mathbf{H}^{\boldsymbol{T}}\mathbf{H}\right)^{-\mathbf{1}}\mathbf{H}^{\boldsymbol{T}}\mathbf{T}$$

- **Equivalent ELM optimization formula**

Minimize: $L_{P_{ELM}} = \frac{1}{2}\|\boldsymbol{\beta}\|^{\mathbf{2}} + C\frac{1}{2}\sum_{i=1}^{N}\|\boldsymbol{\xi}_i\|^2$

subject to: $\boldsymbol{h}(\mathbf{x}_i)\boldsymbol{\beta} = \boldsymbol{t}_i^T + \boldsymbol{\xi}_i^T, \forall i$

Extreme Learning Machines (ELM)

- **Valid for both kernel and non-kernel learning**
  - Non-kernel based:

$$\boldsymbol{f}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\mathbf{H}^{\boldsymbol{T}}\left(\frac{I}{C} + \mathbf{H}\mathbf{H}^{\boldsymbol{T}}\right)^{-\mathbf{1}}\mathbf{T}$$

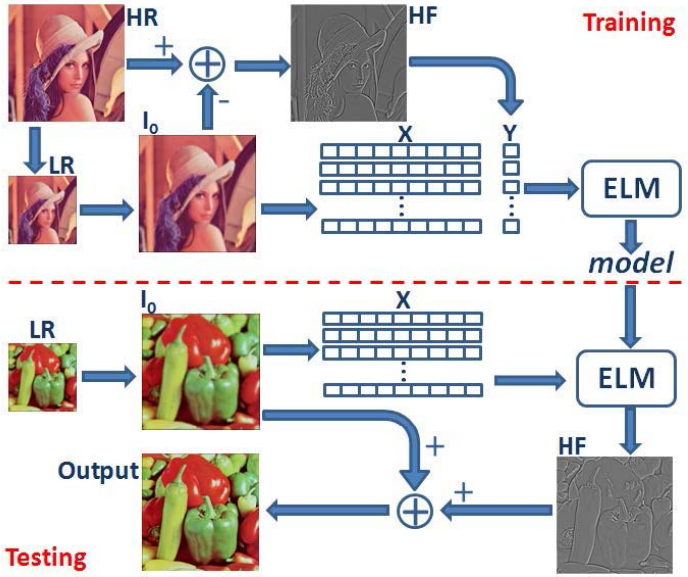
and

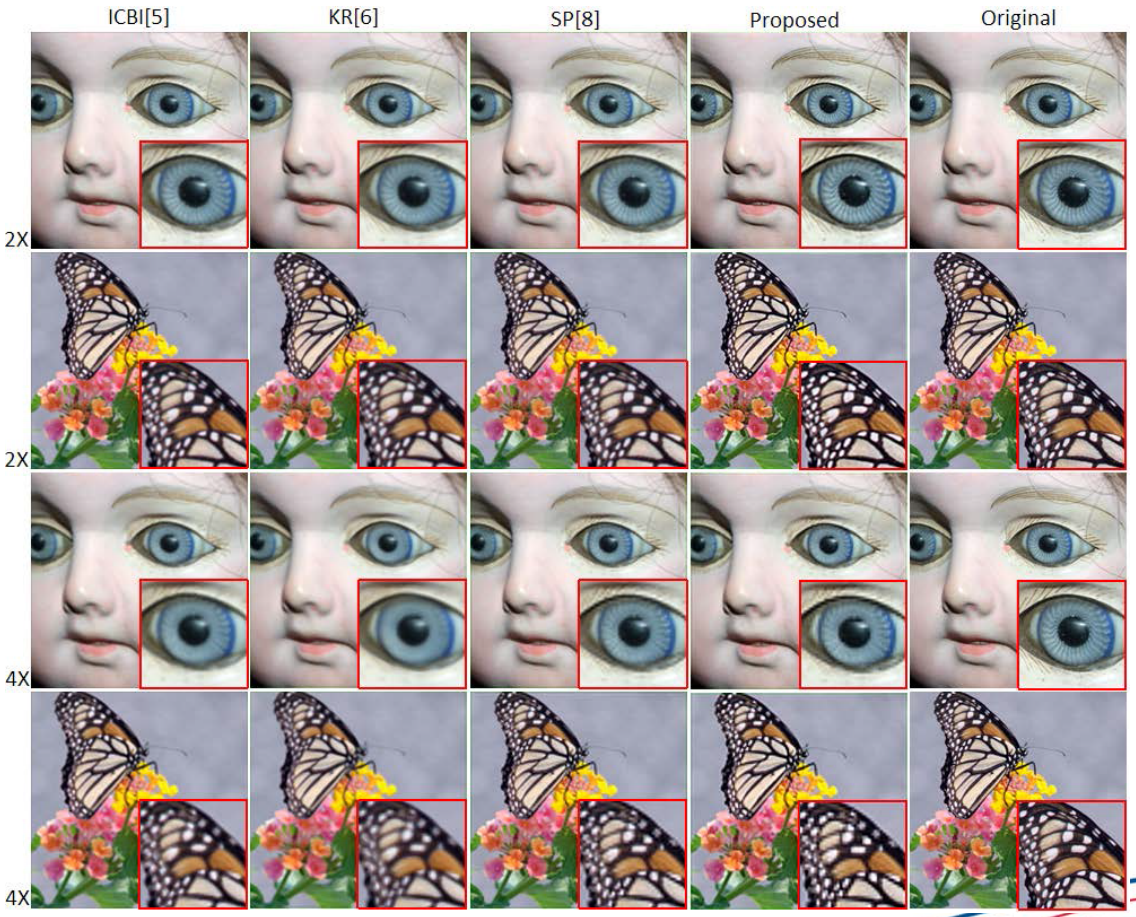
$$\boldsymbol{f}(\mathbf{x}) = \boldsymbol{h}(\mathbf{x})\left(\frac{I}{C} + \mathbf{H}^{\boldsymbol{T}}\mathbf{H}\right)^{-\mathbf{1}}\mathbf{H}^{\boldsymbol{T}}\mathbf{T}$$

  - Kernel based (if $h(\mathbf{x})$ is unknown): $f(\mathbf{x}) = \begin{bmatrix} K(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ K(\mathbf{x}, \mathbf{x}_N) \end{bmatrix}^T \left(\frac{I}{C} + \boldsymbol{\Omega}_{ELM}\right)^{-1}\mathbf{T}$

    where $\boldsymbol{\Omega}_{ELM_{i,j}} = \boldsymbol{h}(\mathrm{x}_i) \cdot \boldsymbol{h}(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$

Image Super-Resolution by ELM





From top to down: super-resolution at 2x and 4x. State-of-the-art methods: iterative curve based interpolation (ICBI), kernel regression based method (KR), compressive sensing based sparse representation method (SR). [An and Bhanu 2012]

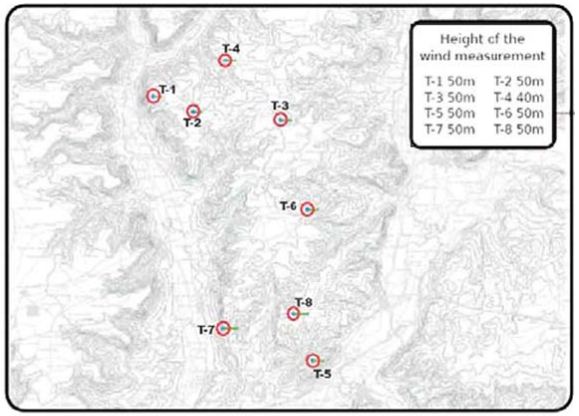
Automatic Object Recognition

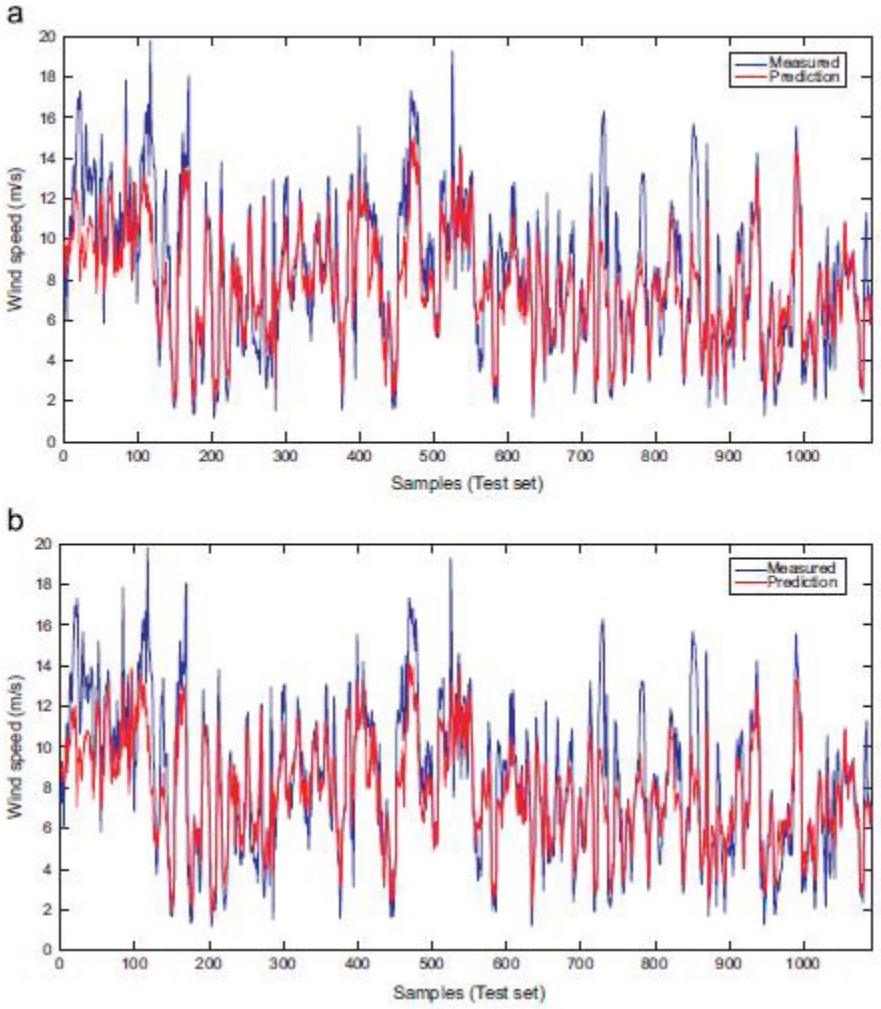
Object Categories	ELM Based	AdaBoost Based	Joint Boosting	Scale-Invariant Learning
Bikes	94.6	93.4	92.5	73.9
Planes	95.3	90.0	90.2	92.7
Cars	99.0	96.0	90.3	97.0
Leaves	98.3	94.2	-	97.8
Faces	97.9	98.0	96.4	-

[Minhas, et al 2010]

Sample images from CalTech database

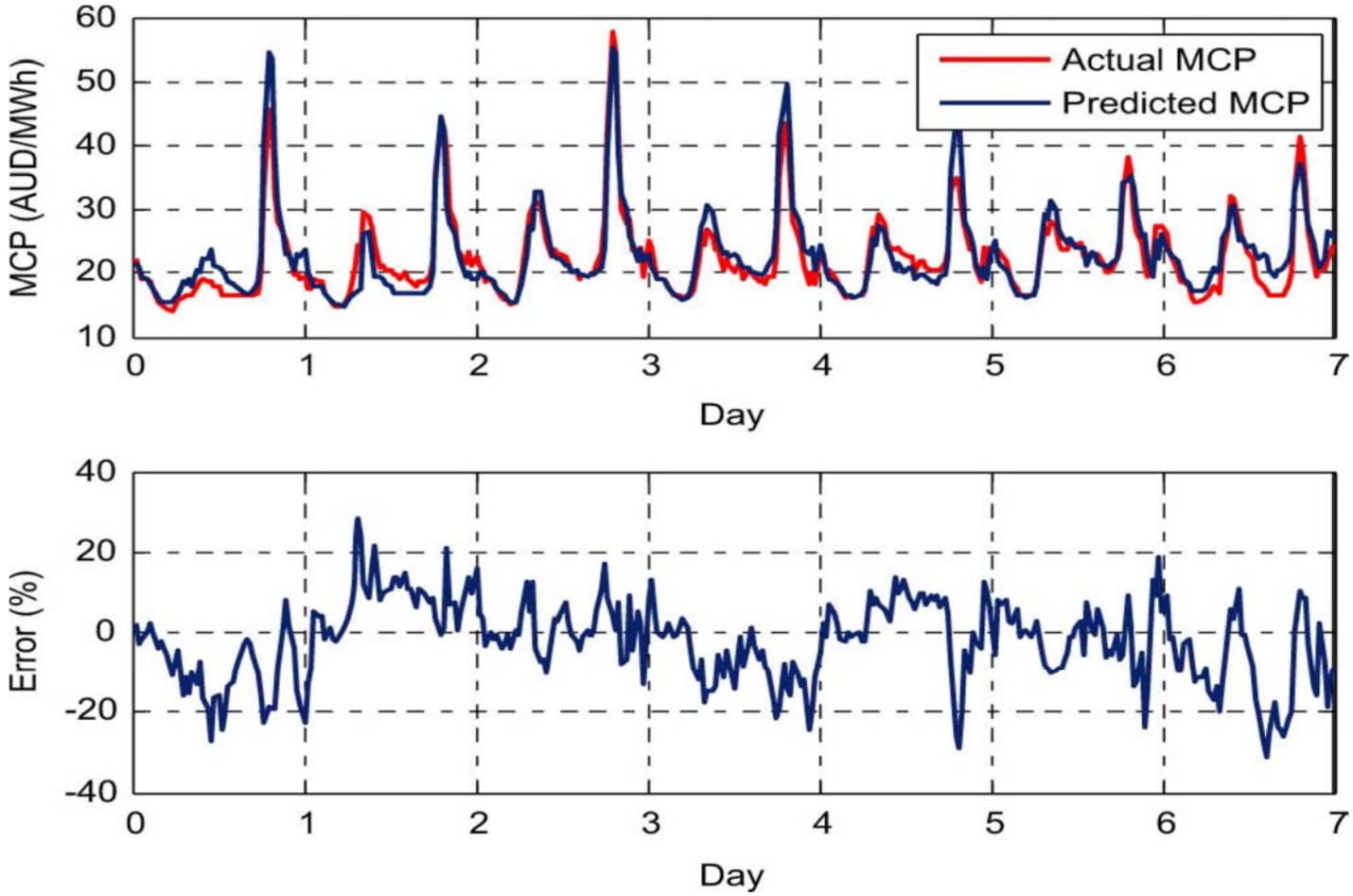
Real Operation of Wind Farms





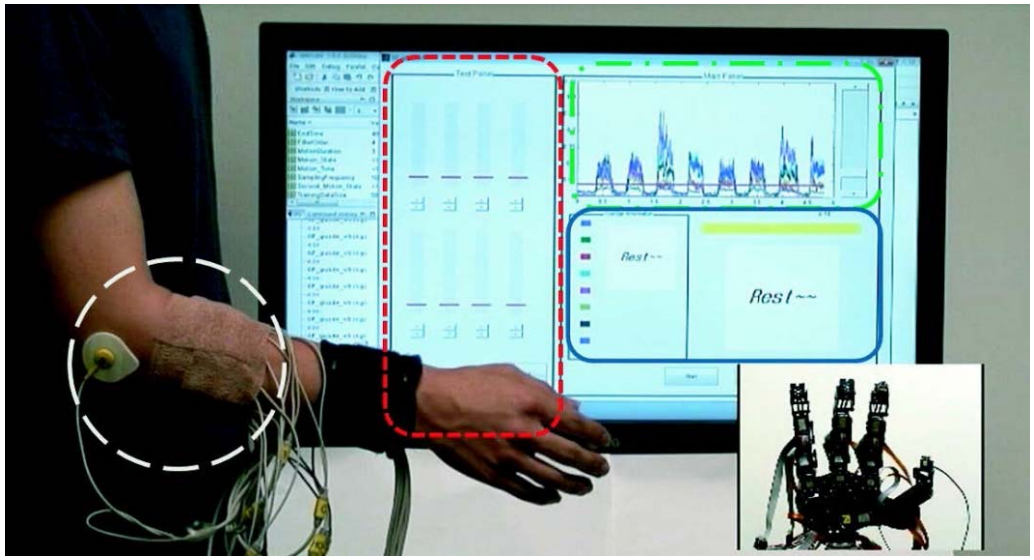
Situation of the wind measuring towers in Spain and within the eight wind farms. Wind speed prediction in tower 6 of the considered wind farm in Spain obtained by the ELM network (prediction using data from 7 towers). (a) Best prediction obtained and (b) worst prediction obtained. [Saavedra-Moreno, et al, 2013]

Electricity Price Forecasting

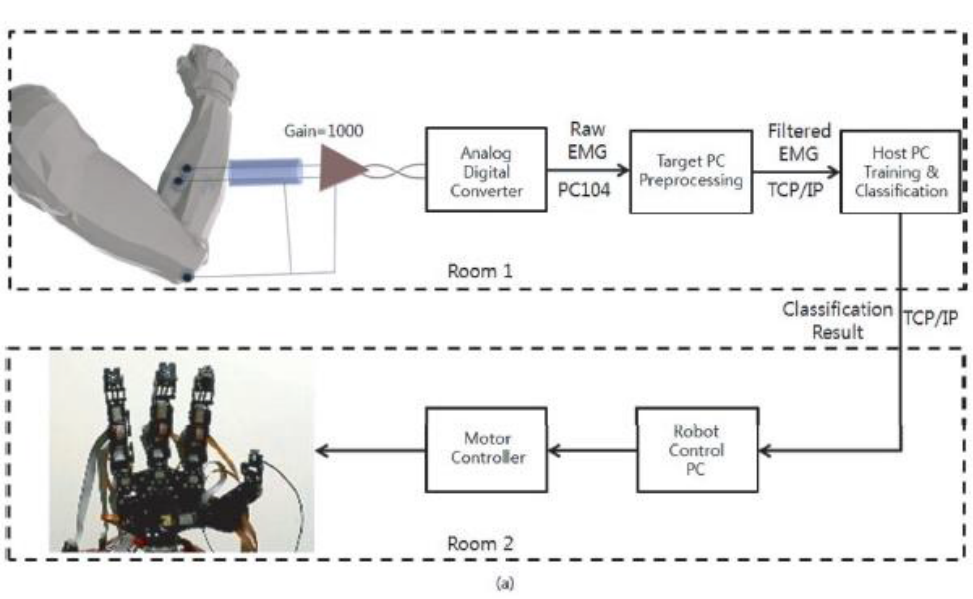


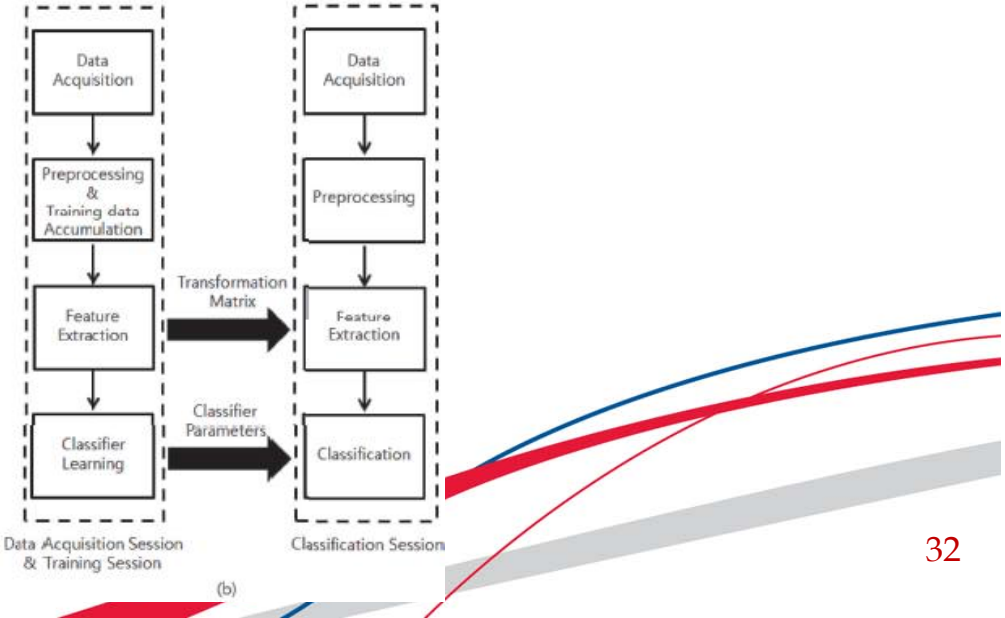
Average results of market clearing prices (MCP) forecast by ELM in winter: Trading in the Australian national electricity market (NEM) is based on a 30-min trading interval. Generators submit their offers every 5 min each day. Dispatch price is determined every 5 min and 6 dispatch prices are averaged every half-hour to determine the regional MCPs. In order to assist decision-making process for generators, there are totally 48 MCPs needed to be predicted at the same time for the coming trading day. [Chen, et al, 2012]

Remote Control of a Robotic Hand



- An eight wrist motions offline classification using linear support vector machines with little training time (under 10 minutes).
- This study shows human could control the remote side robot hand in real-time using his or her sEMG signals with less than 50 seconds recorded training data with ELM.[Lee, et al 2011]



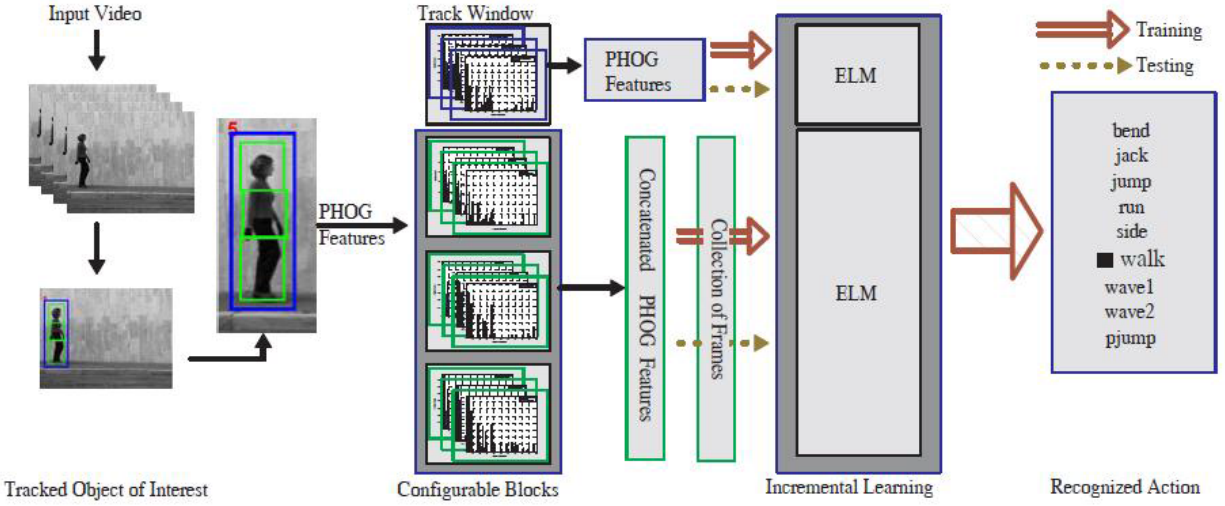


Human Action Recognition

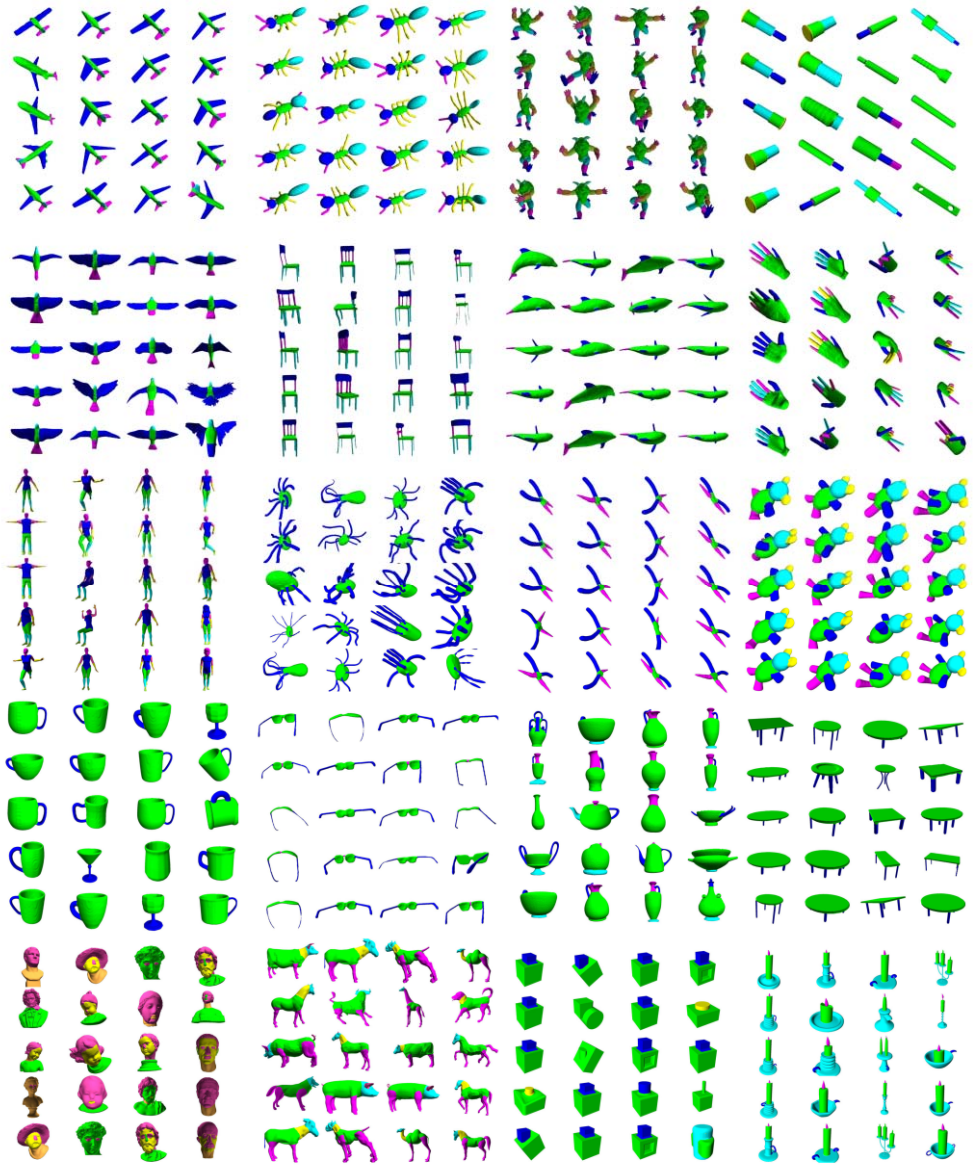
Weizmann dataset											
Methods	OS-ELM Based				[2]	[32]	[14]	[36]	[41]	[30]	[11]
Frames	1/1	3/3	6/6	10/10	-	-	-	-	-	-	-
Accuracy	100.0	100.0	100.0	100.0	100.0	72.8	98.8	100.0	97.8	99.44	100.0

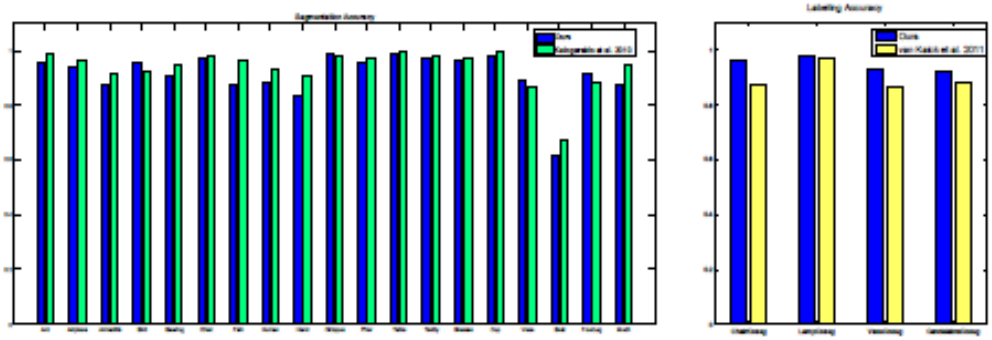
KTH dataset											
Methods	OS-ELM Based				[14]	[36]	[30]	[21]	[27]	[9]	[44]
Frames	1/1	3/3	6/6	10/10	-	-	-	-	-	-	-
Accuracy	92.8	93.5	95.7	96.1	91.7	92.7	94.83	95.77	97.0	96.7	95.7

[Minhas, et al 2012]



3D Shape Segmentation and Labelling





Training time shortened from **8 hours** (conventional methods) to **15 seconds** (ELM solution) for a dataset with 6 meshes with about 25~30K faces.

[Xie, et al 2014]

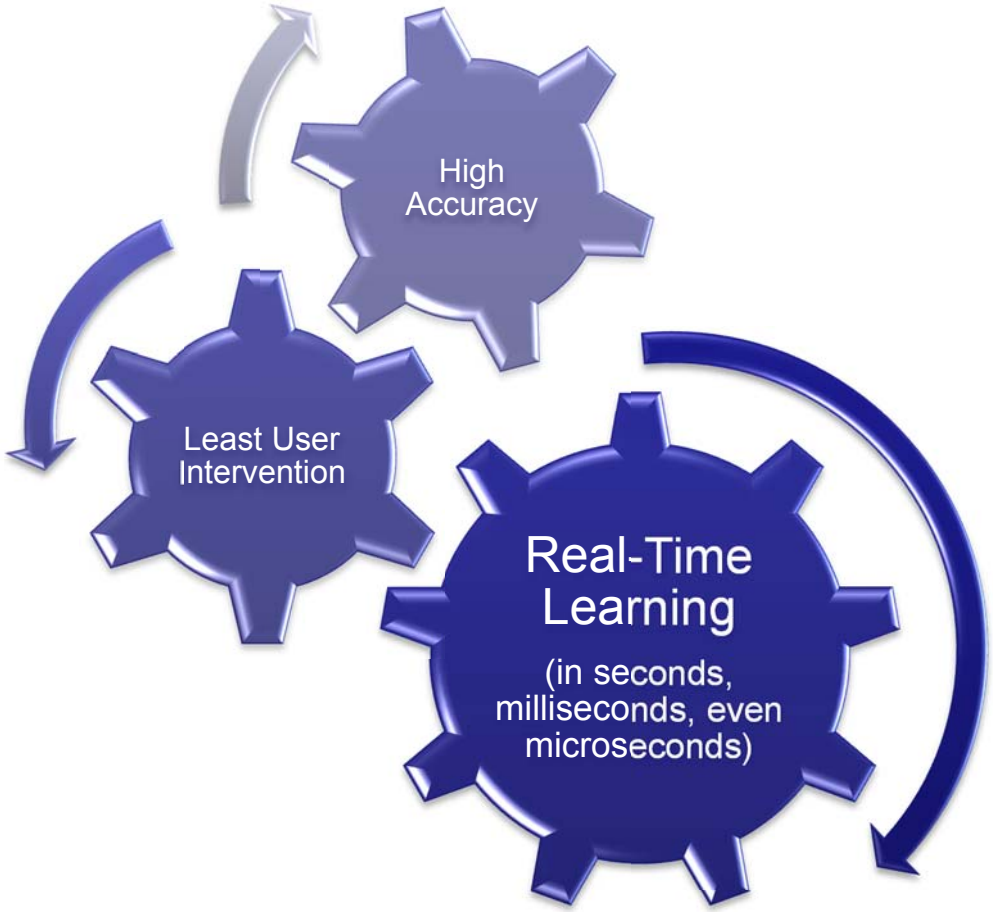
Constraints of BP and SVM Theory

- **Both PDP Group and V. Vapnik have made great contributions in neural networks R&D**
  - Without PDP Group's work on BP in 1986, neural networks might not have revived in 1980's.
  - Without Vapnik's work on SVM in 1995, neural networks might have disappeared although many SVM researchers do not consider SVM a kind of solutions to the traditional neural networks.
  - Without SVM, many applications in pattern recognition, HCI, BCI, computational intelligence and machine learning, etc, may not have appeared and been so successful.
- **However, …**

Constraints of BP and SVM Theory

- **However, both BP and SVM over-emphasize some aspects of learning and overlook the other aspects, and thus, both become incomplete in theory:**
  - BP gives preference on training but does not consider the stability of the system (consistency of minimum norm of weights in neural networks, linear system, and matrix theory)
  - SVM confines the research in the maximum margin concept which limits the research in binary classification and does not have direct and efficient solutions to regression and multi-class applications. The consistency between maximum margin, minimum norm of weights in neural networks and matrix theory has been overlooked.

Essential Considerations of ELM



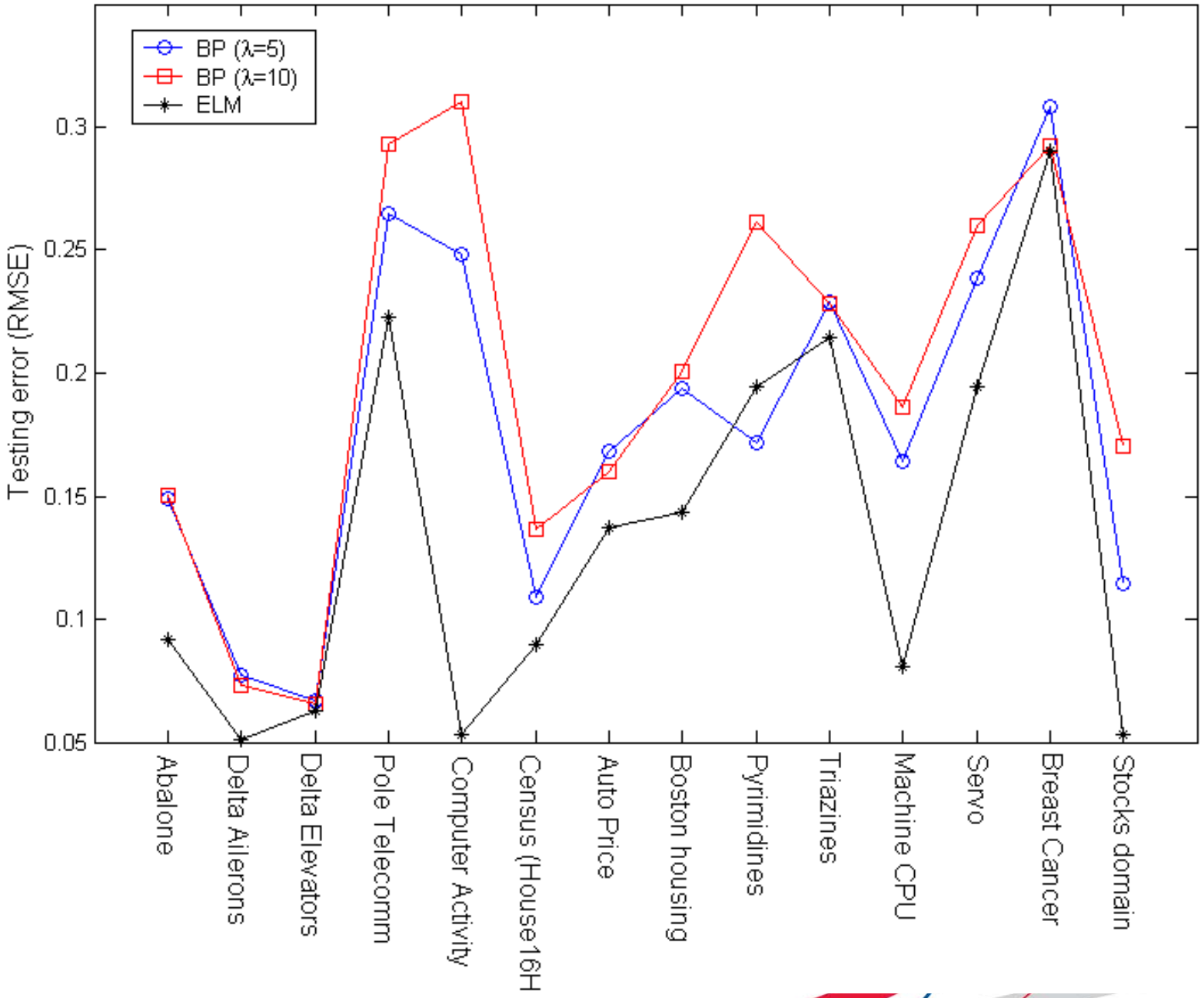
ELM for Threshold Networks

- Binary / Threshold node:

$$g(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

- Threshold networks (in fact approximated by sigmoid networks in literature) were usually trained by BP and its variants indirectly in the past three decades. There was no direct learning solution to threshold networks in the past 60 years.
- Threshold unit can be approximated by sigmoid unit $g(x) = 1/(1 + \exp(-\lambda x))$ with sufficiently large gain parameter λ
- With ELM, threshold networks can be trained directly.

ELM for Threshold Networks



ELM for Complex Networks

- Circular functions:
  - $\tan(z) = \frac{e^{iz}-e^{-iz}}{i(e^{iz}+e^{-iz})}$, $\sin(z) = \frac{e^{iz}-e^{-iz}}{2i}$
- Inverse circular functions:
  - $\arctan(z) = \int_0^z \frac{dt}{1+t^2}$, $\arcsin(z) = \int_0^z \frac{dt}{(1-t)^{1/2}}$, $\arccos(z) = \int_0^z \frac{dt}{(1-t^2)^{1/2}}$
- Hyperbolic functions:
  - $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$, $\sinh(z) = \frac{e^{z}-e^{-z}}{2}$
- Inverse hyperbolic functions:
  - $\operatorname{arctanh}(z) = \int_0^z \frac{dt}{1-t^2}$, $\operatorname{arcsinh}(z) = \int_0^z \frac{dt}{(1+t^2)^{1/2}}$

ELM for Complex Networks

- **Wireless Communication Channel Equalizer**
  - Channel model with nonlinear distortion for 4-QAM signals.

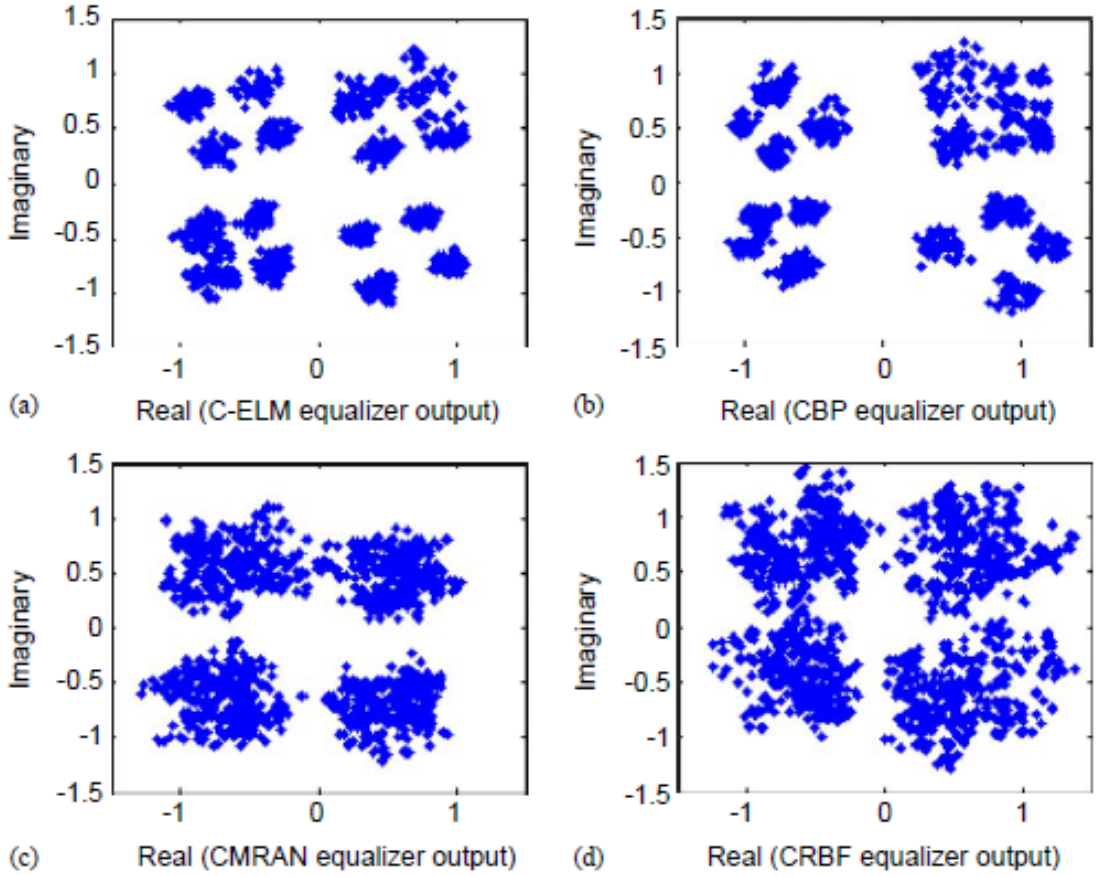
$$z_n = o_n + 0.1o_n^2 + 0.05o_n^3 + v_n, v_n \sim N(0, 0.01)$$

$$o_n = (0.34 - i0.27)s_n + (0.87 + i0.43)s_{n-1} + (0.34 - i0.21)s_{n-2}$$

Complex activation function used in ELM: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, where $z = \mathbf{a} \cdot \mathbf{z} + b$

ELM for Complex Networks

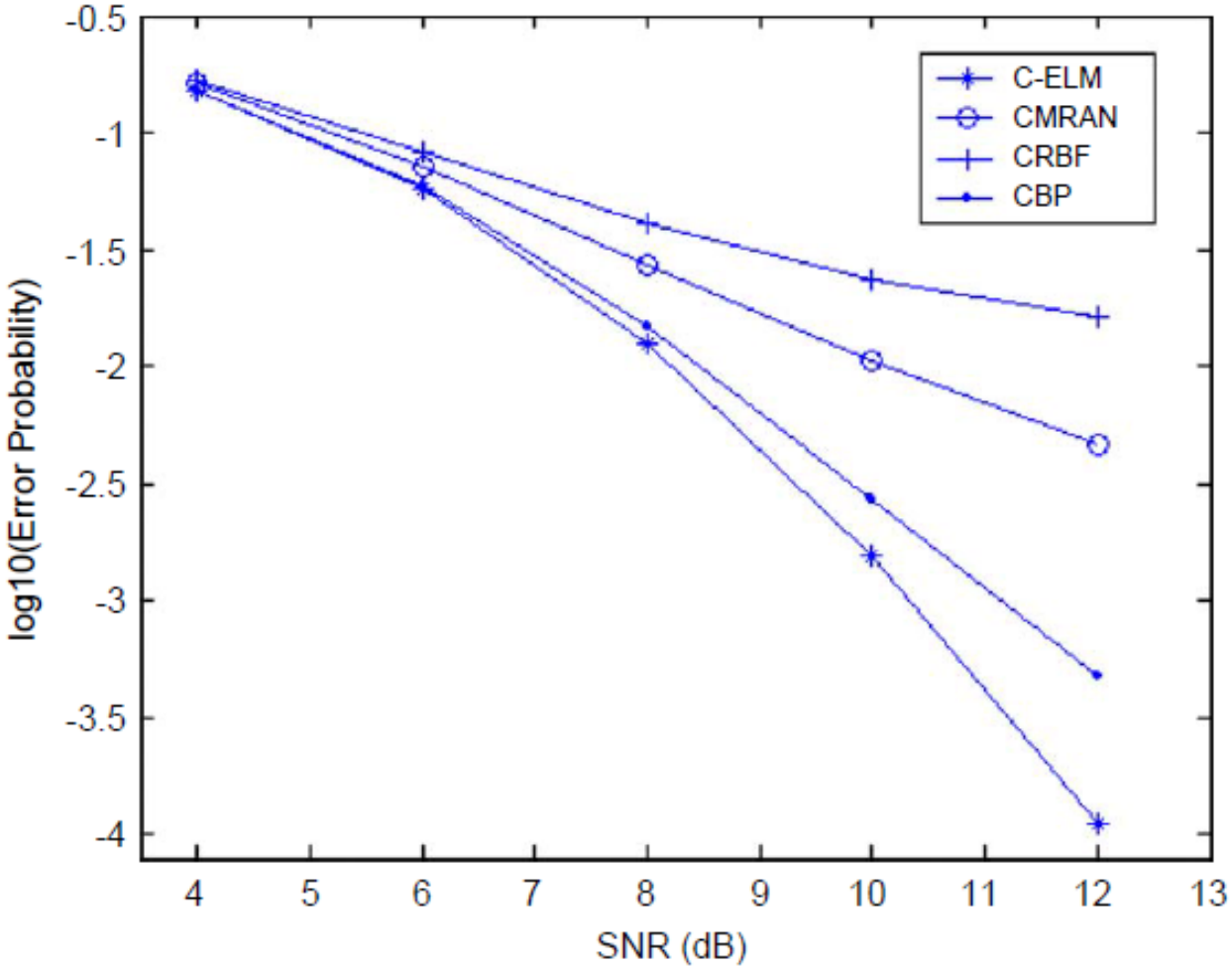
- **Wireless Communication Channel Equalizer**
  - Channel model with nonlinear distortion for 4-QAM signals.

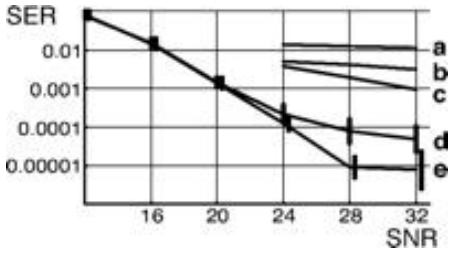


Eye diagram of the outputs of different equalizers (a) C-ELM (ELM with complex hidden nodes), (b) CBP (Complex valued BP), (c) CMRAN (Complex valued MRAN), (d) CRBF (Complex valued RBF).

ELM for Complex Networks

- **Save Energy in Wireless Communication**



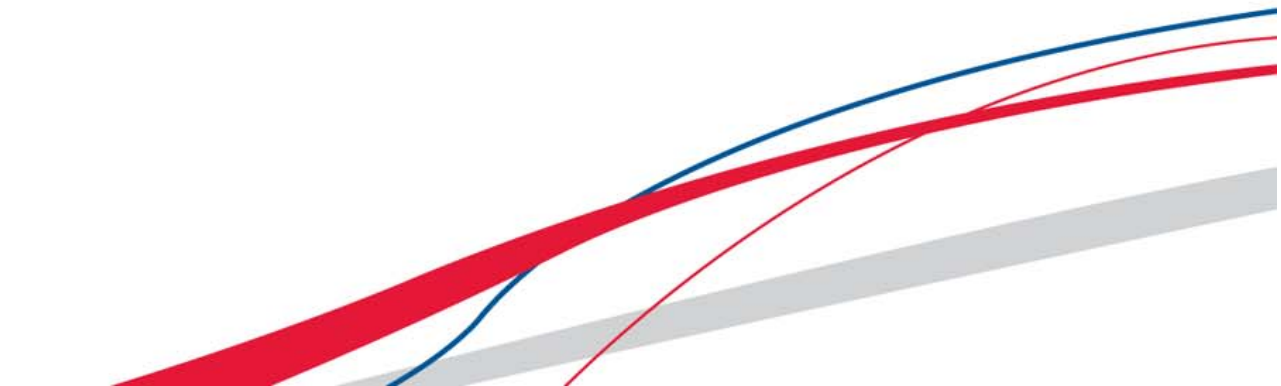


SER versus SNR: (a) Linear DFE. (b) Volterra DFE. (c) Bilinear DFE. (d) represents average ESN (Echo State Network) performance with randomly generated reservoirs. (e) indicates performance of best network chosen from the networks averaged in (d). From H. Jaeger and H. Haas, *Science*, vol. 404, pp. 78-80, 2004.

Compared with ESN, ELM reduces the error rate by 1000 times or above.

NANYANG TECHNOLOGICAL UNIVERSITY

Why SVM / LS-SVM Are Suboptimal



Optimization Constraints of ELM and LS-SVM

- **ELM: Based on Equality Constraint Conditions** [Huang, et al 2012]
  - ELM optimization formula:

    Minimize: $L_{P_{ELM}} = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\|\boldsymbol{\xi}_i\|^2$

    subject to: $\boldsymbol{h}(\mathbf{x}_i)\boldsymbol{\beta} = \boldsymbol{t}_i^T + \boldsymbol{\xi}_i^T, \forall i$
  - The corresponding dual optimization problem:

    Minimize: $L_{D_{ELM}} = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\|\boldsymbol{\xi}_i\|^2 - \sum_{i=1}^{N}\sum_{j=1}^{m}\left(\boldsymbol{h}(\mathbf{x}_i)\boldsymbol{\beta} - \boldsymbol{t}_i^T + \boldsymbol{\xi}_i^T\right)\boldsymbol{\alpha}_i$

    subject to: $\boldsymbol{\beta} = \mathbf{H}^{\boldsymbol{T}}\boldsymbol{\alpha}, \boldsymbol{\alpha}_i = C\boldsymbol{\xi}_i, \boldsymbol{h}(\mathbf{x}_i)\boldsymbol{\beta} - \boldsymbol{t}_i^T + \boldsymbol{\xi}_i^T = 0, \forall i$

Optimization Constraints of ELM and LS-SVM

- **LS-SVM: Based on Equality Constraint Conditions** [Suykens and Vandewalle 1999]
  - LS-SVM optimization formula:

    Minimize: $L_{P_{LS-SVM}} = \frac{1}{2}\|\mathbf{w}\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\xi_i^2$

    subject to: $t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b) = 1-\xi_i \ , \forall i$

  - The corresponding dual optimization problem:

    Minimize: $L_{D_{LS-SVM}} = \frac{1}{2}\|\mathbf{w}\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\xi_i^2 - \sum_{i=1}^{N}\alpha_i\big(t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b) - 1+\xi_i\ \big)$

    subject to:

    $\mathbf{w} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\alpha_i t_i\phi(\mathbf{x}_i)\,, \alpha_i = C\xi_i, t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b) - 1+\xi_i\ = 0, \forall i$

    $\sum_{i=1}^{N}\alpha_i t_i = 0$

In LS-SVM optimal α_i are found from one hyper plane $\sum_{i=1}^{N}\alpha_i t_i = 0$

Optimization Constraints of ELM and SVM

- **ELM: Based on Inequality Constraint Conditions** [Huang, et al 2010]
  - ELM optimization formula:

Minimize: $L_{P_{ELM}} = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\sum_{i=1}^{N}\xi_i$

subject to: $t_i \boldsymbol{h}(\mathbf{x}_i)\boldsymbol{\beta} \geq 1 - \xi_i \ , \forall i$

$\xi_i \geq 0, \forall i$

  - The corresponding dual optimization problem:

Minimize: $L_{D_{ELM}} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} t_i t_j \alpha_i \alpha_j \, \boldsymbol{h}(\mathbf{x}_i) \cdot \boldsymbol{h}(\mathbf{x}_j) - \sum_{i=1}^{N}\alpha_i$

subject to: $0 \leq \alpha_i \leq C, \forall i$

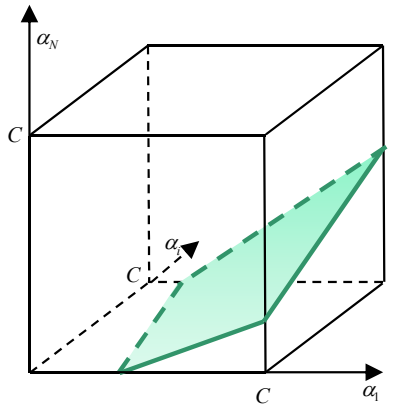
Optimization Constraints of ELM and SVM

- **SVM: Based on Inequality Constraint Conditions** [Cortes and Vapnik 1995]
  - SVM optimization formula:

Minimize: $L_{P_{SVM}} = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$

subject to: $t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \ , \forall i$

$\xi_i \geq 0, \forall i$



In SVM optimal α_i are found from one hyper plane $\sum_{i=1}^{N}\alpha_i t_i = 0$

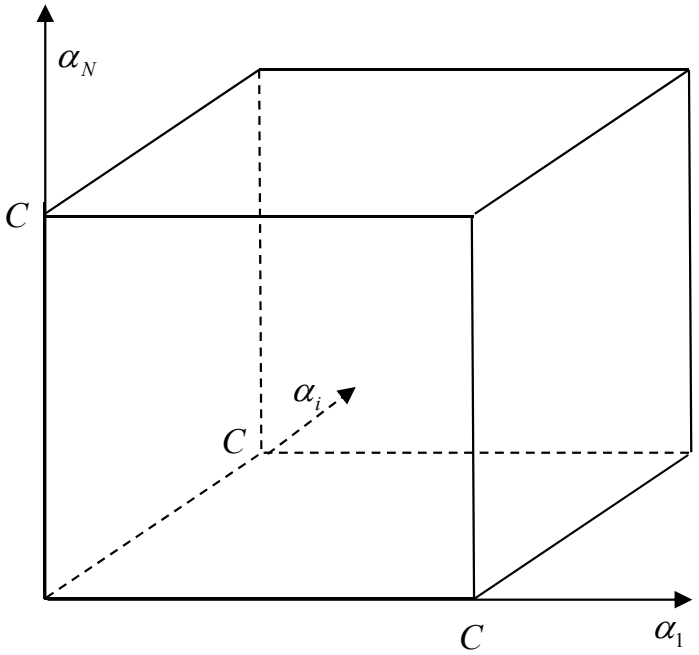
  - The corresponding dual optimization problem:

Minimize: $L_{D_{SVM}} = \sum_{i=1}^{N}\sum_{j=1}^{N} t_i t_j \alpha_i \alpha_j \, \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) - \sum_{i=1}^{N}\alpha_i$

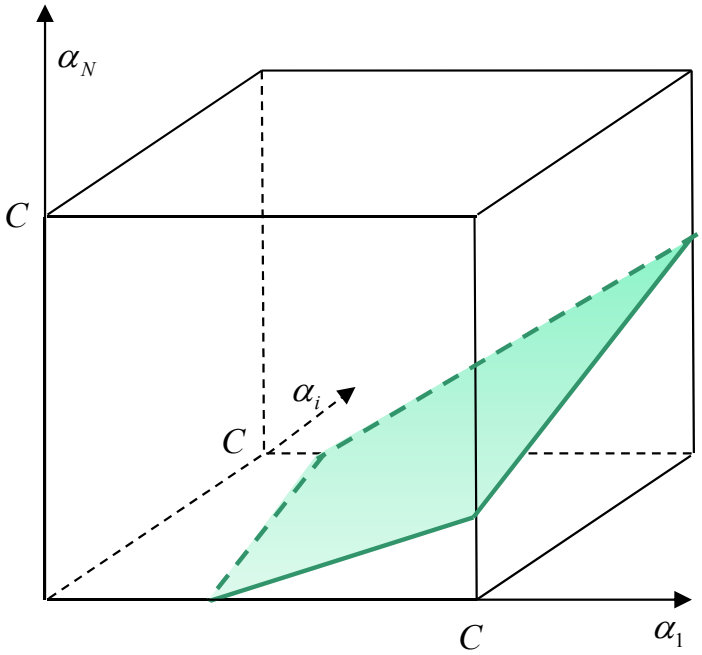
subject to: $0 \leq \alpha_i \leq C, \forall i$

$\sum_{i=1}^{N}\alpha_i t_i = 0$

Optimization Constraints of ELM and SVM



ELM's inequality constraint variant [Huang, et al 2010]



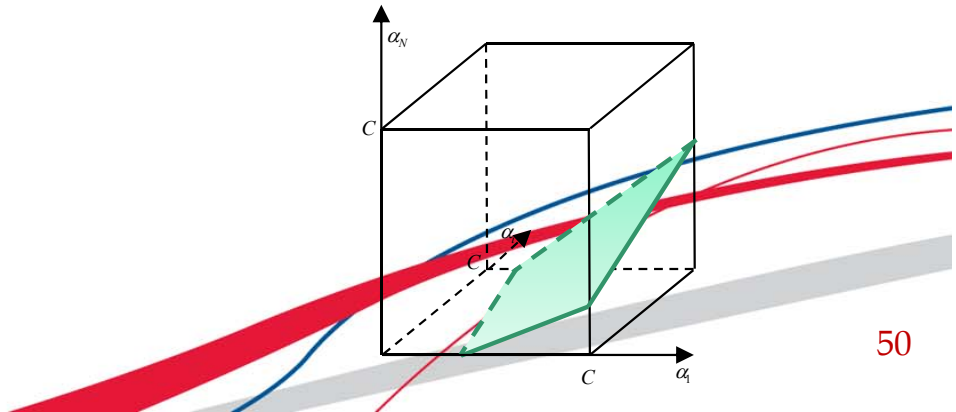
SVM

ELM (based on inequality constraint conditions) and SVM have the same dual optimization objective functions, but in ELM optimal α_i are found from the entire cube $[0, C]^N$ while in SVM optimal α_i are found from one hyperplane $\sum_{i=1}^{N} \alpha_i t_i = 0$ within the cube $[0, C]^N$. SVM always provides a suboptimal solution, so does LS-SVM.

SVM's Suboptimal Solutions

- **Reasons**
  - *SVM's historical role is irreplaceable! Without SVM and Vapnik, computational intelligence may not be so successful and the history of computational intelligence would be re-written!* However ...
  - SVM always searches for the optimal solution in the hyperplane $\sum_{i=1}^{N} \alpha_i t_i = 0$ within the cube $[0, C]^N$ of the SVM feature space.
  - SVMs may apply similar application-oriented constraints to irrelevant applications and search similar hyper planes in feature space if their target labels are similar. Irrelevant applications may become relevant in SVM solutions.

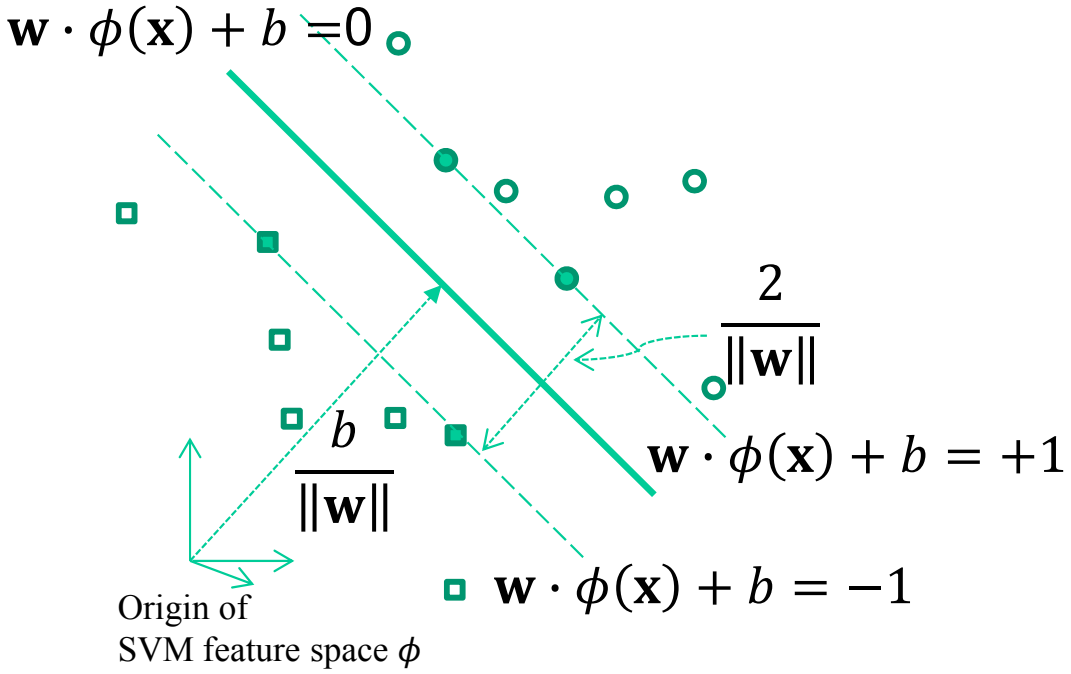
[Huang, et al 2010]

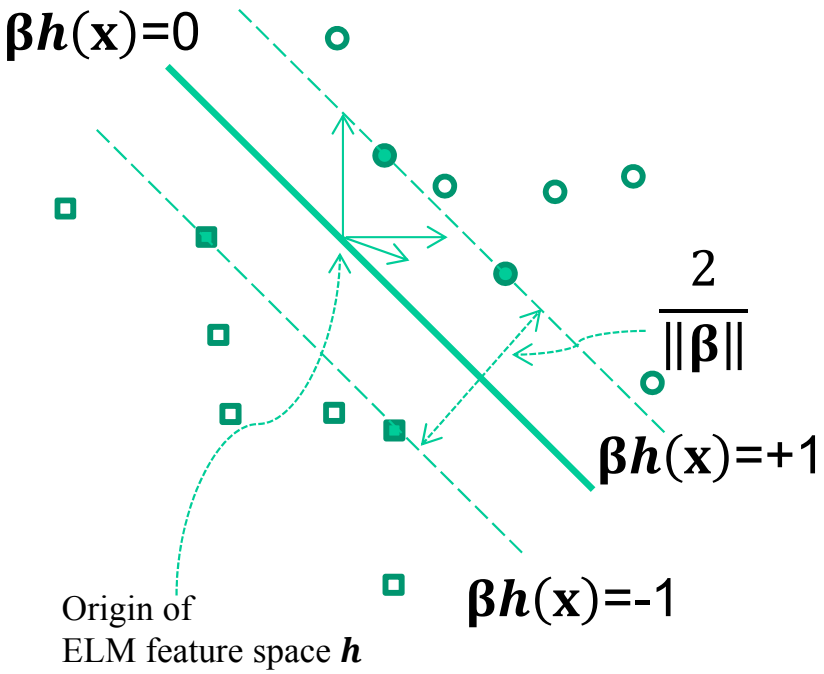


SVM's Suboptimal Solutions

- **Reasons**
  - SVM is too "generous" on the feature mappings and kernels, almost condition free except for Mercer's conditions.
    1) As the feature mappings and kernels need not satisfy universal approximation condition, b must be present.
    2) As b exists, contradictions are caused.
    3) LS-SVM inherits such "generosity" from the conventional SVM

SVM's Suboptimal Solutions





As SVM was originally proposed for classification, universal approximation capability was not considered at the first place. Actually the feature mappings $\phi(\mathbf{x})$ are unknown and may not satisfy universal approximation condition, b must be present to absorb the system error. ELM was originally proposed for regression, the feature mappings $\boldsymbol{h}(\mathbf{x})$ are known and universal approximation capability was considered at the first place. In ELM the system error tends to be zero and b should not be present.

SVM's Suboptimal Solutions

- **Maximum margin?**
  - Maximum margin is good to binary classification cases. However, if only considering maximum margin, one may not be able to imagine "maximum margin" in multi-class / regression problems.
  - To over-emphasize "maximum margin" makes the SVM research deadlock in binary classification and difficult to find the direct solution to multi-class applications
  - "Maximum margin" is just a special case of ridge regression theory, linear system stability, and neural network generalization performance theory in binary applications.
- **ELM integrates the ridge regression theory, linear system stability, and neural network generalization performance theory for regression and multiclass applications, "maximum margin" is just a special case in ELM's binary applications.**

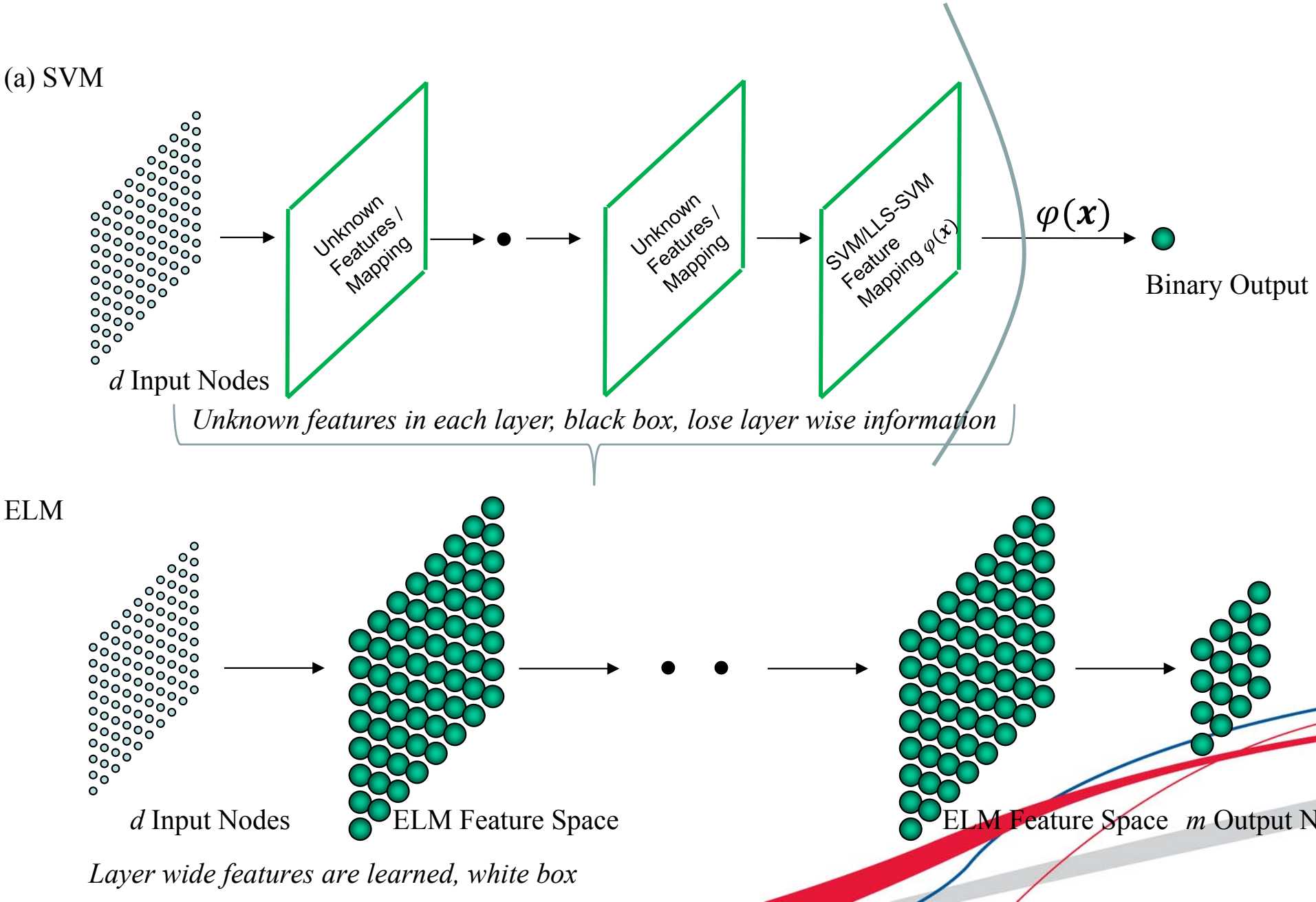
SVM's Suboptimal Solutions

- **Data distortion in multi-class classifications?**
  - Different from ELM, SVM and LS-SVM do not have direct solutions to multi-class applications. Usually SVM and LS-SVM use One-Against-One (OAO) or One-Against-All (OAA) methods to handle multi-class applications indirectly, which may distort applications.

G.-B. Huang, et al., "Extreme learning machine for regression and multiclass classification", *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 42, no. 2, pp. 513-529, 2012.

G.-B. Huang, "An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels", *Cognitive Computation*, 2014.

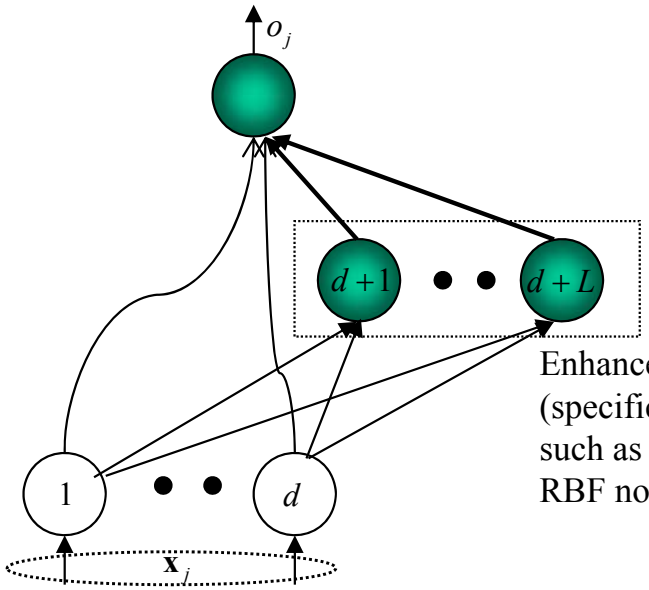
ELM and SVM


(a) SVM
Unknown Features / Mapping
Unknown Features / Mapping
SVM/LLS-SVM Feature Mapping $\varphi(x)$
$\varphi(x)$
Binary Output
d Input Nodes
Unknown features in each layer, black box, lose layer wise information
ELM
d Input Nodes
ELM Feature Space
ELM Feature Space
m Output Nodes
Layer wide features are learned, white box

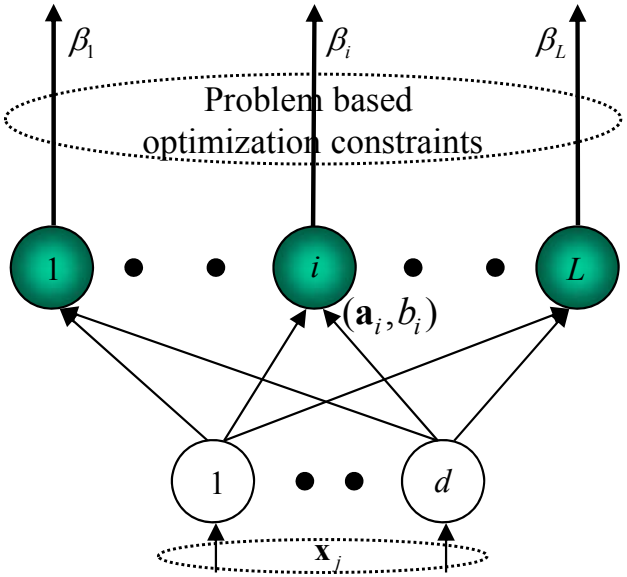
Relationship and Difference Between ELM and SVM/LS-SVM

Properties	ELMs	SVM	LS-SVM
Belief	Unlike conventional learning theories and common understanding, ELM belief: Learning can be made without tuning hidden neurons in wide type of biological learning mechanisms and wide types of neural networks	No such belief (Original assumption [15]: If there is no learning solution for feedforward networks, one only needs to consider the output of the last hidden layer: $\phi(\mathbf{x})$)	No such belief (Original assumption [15]: If there is no learning solution for feedforward networks, one only needs to consider the output of the last hidden layer: $\phi(\mathbf{x})$)
Biological inspired	Yes (Confirmed in rats' olfactory system / visual system)	No	No
Network output functions	$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x})$	$f(\mathbf{x}) = \sum_{s=1}^{N_s} \alpha_s t_s \phi(\mathbf{x}) \cdot \phi(\mathbf{x}_s) + b$	$f(\mathbf{x}) = \sum_{s=1}^{N_s} \alpha_s t_s \phi(\mathbf{x}) \cdot \phi(\mathbf{x}_s) + b$
Multiclass classification	Direct solutions	Indirect solutions based on binary ($t_i = 0$ or 1) case	Indirect solutions based on binary ($t_i = 0$ or 1) case
Explicit feature mappings	Yes (Wide types of explicit feature mappings $\mathbf{h}(\mathbf{x})$. Kernels can also be used.)	No (Unknown mapping $\phi(\mathbf{x})$, kernel only.)	No (Unknown mapping $\phi(\mathbf{x})$, kernel only.)
Hidden node types (mathematical model)	Wide types (sigmoid, kernel, Fourier series, etc)	Kernels	Kernels
Hidden node types (biological neurons)	Yes	No	No
Domain	Both real and complex domains	Real domain (Difficult in handling complex domain directly)	Real domain (Difficult in handling complex domain directly)
SLFNs	"Generalized" SLFN Wide types of SLFNs	No	No
Layer wise feature representation	Yes	No (Feature representations in different layers are ignored)	No (Feature representations in different layers are ignored)
Connectivity	For both fully connected and randomly (partially) connected network	No attention on network connections	No attention on network connections
Hyperplane constraints in dual problem	No (It has no such hyperplane constraints due to lack of bias b in output nodes.)	Yes (It has such hyperplane constraints due to bias b in output nodes.)	Yes (It also provides the model without b but it does still assume binary class. [45])
Universal approximation and classification capability	Proved theoretically for wide types of nonlinear piecewise nodes / neurons	No theoretical proof	No theoretical proof
Ridge regression theory	Yes (Consistent for feature learning, clustering, regression and binary / multiclass classification.)	No (Maximal margin concept is a specific case of ridge regression theory used in binary classification.)	No (Maximal margin concept is a specific case of ridge regression theory used in binary classification.)
Learning capability	Efficient in feature learning (auto-encoders) and clustering	Difficult in handling auto-encoders	Difficult in handling auto-encoders
Solutions	Closed-form and non-closed-form, online, sequential and incremental	Non-closed-form	Closed-form

ELM vs QuickNet / RVFL



Enhanced Patterns
(specific ELM feature mapping such as sigmoid nodes and RBF nodes)



QuickNet (1989, not patented) / **RVFL** (1994, patented)	**ELM** (not patented)
$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i G_{sig,RBF} + \boldsymbol{\alpha} \cdot \mathbf{x}$	$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x})$
Mainly on sigmoid and RBF nodes, not applicable to kernels learning	Proved on general cases: any piecewise continuous nodes. ELM theories extended to biological neurons whose mathematical formula is even unknown
Not feasible for multi-layer of RVFL, losing learning in auto-encoder and feature learning. RVFL and PCA/Random project are different	Efficient for multi-layer of ELM, auto-encoder, and feature learning, PCA and random projects are specific cases of ELM when linear neurons are used.
If ELM's optimization is used in QuickNet (1988) / RVFL and Schimidt (1992), a suboptimal solution tends to be achieved.	Regularization of output weights, ridge regression theories, neural networks generalization performance theories (maximal margin in binary class cases), SVM and LS-SVM provide suboptimal solutions.
Hidden layer output matrix: [$\mathbf{H}_{\text{ELM for Sig or RBF}}$, $\mathbf{X}_{N \times d}$]	Hidden layer output matrix: $\mathbf{H}_{\text{ELM for almost any nonlinear piecewise neurons}}$
	Homogenous architectures for compression, feature learning, clustering, regression and classification

Relationship and Difference Between ELM and QuickNet/RVFL, Duin's Work

Properties	ELMs	Schmidt, *et al.* (1992)	QuickNet / RVFL
Belief	Unlike conventional learning theories and common understanding, ELM belief: Learning can be made without tuning hidden neurons in wide type of biological learning mechanisms and wide types of neural networks	No such belief	No such belief
Network output functions	$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x})$	$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i g_{\text{sig}}(\mathbf{a}_i \cdot \mathbf{x} + b_i) + b$	$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i g_{\text{sig or RBF}} + \alpha \cdot \mathbf{x}$
SLFNs	"Generalized" SLFN in which a hidden node can be a subnetwork	Standard SLFN only	Standard SLFN only
Multi layers networks	Yes	No	No
Connectivity	For both fully connected and randomly (partially) connected network	Fully connected	Fully connected
Hidden node types (mathematical model)	Wide types (sigmoid, kernel, Fourier series, etc)	Sigmoid	Sigmoid and RBF
Hidden node types (biological neurons)	Yes	No	No
Domain	Both real and complex domains	Real domain	Real domain
Hidden layer output matrix	$\mathbf{H}_{\text{ELM}}$	$[\mathbf{H}_{\text{ELM for sigmoid basis}}, \mathbf{1}_{N\times m}]$	$[\mathbf{H}_{\text{ELM for sigmoid or RBF basis}}, \mathbf{X}_{N\times d}]$
Universal approximation and classification capability	Proved for wide types of random neurons	No theoretical proof	Theoretical proof for semi-random sigmoid or RBF nodes
Structural risk minimization	Minimize: $\|\beta\|_p^{\sigma_1} + C\|\mathbf{H}\beta - \mathbf{T}\|_q^{\sigma_2}$	Not considered	Not considered
Learning capability	Efficient in feature learning (auto-encoders) and clustering	Difficult in handling auto-encoders	Lose learning capability in auto-encoders
Solutions	Closed-form and non-closed-form, sequential and incremental	Closed-form	Closed-form and non-closed-form for QuickNet, Closed-form for RVFL
Portability	Many (*but not all*) ELM variants can be linearly extended to Schmidt, *et al.* (1992) and RVFL/QuickNet instead of vice versa, the resultant algorithms are referred to as "ELMs+b" for Schmidt, *et al.* (1992) and "ELM+αx" for QuickNet/RVFL		

G.-B. Huang, "What are Extreme Learning Machines? Filling the Gap between Frank Rosenblatt's Dream and John von Neumann's Puzzle", *Cognitive Computation*, vol. 7, pp. 263-278, 2015.

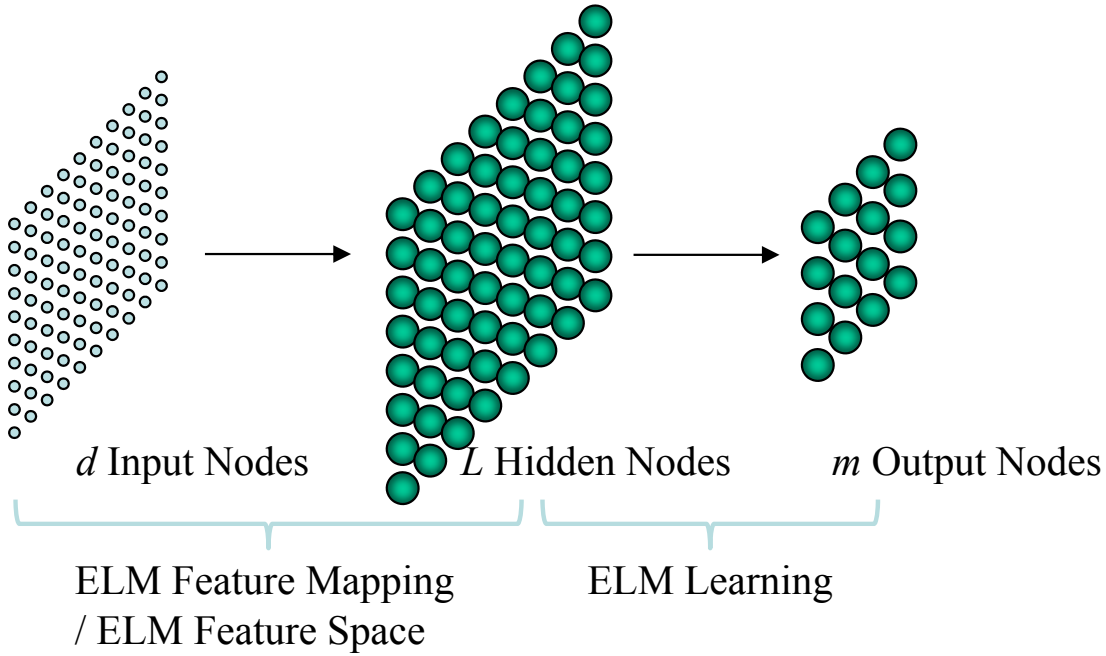
NANYANG TECHNOLOGICAL UNIVERSITY

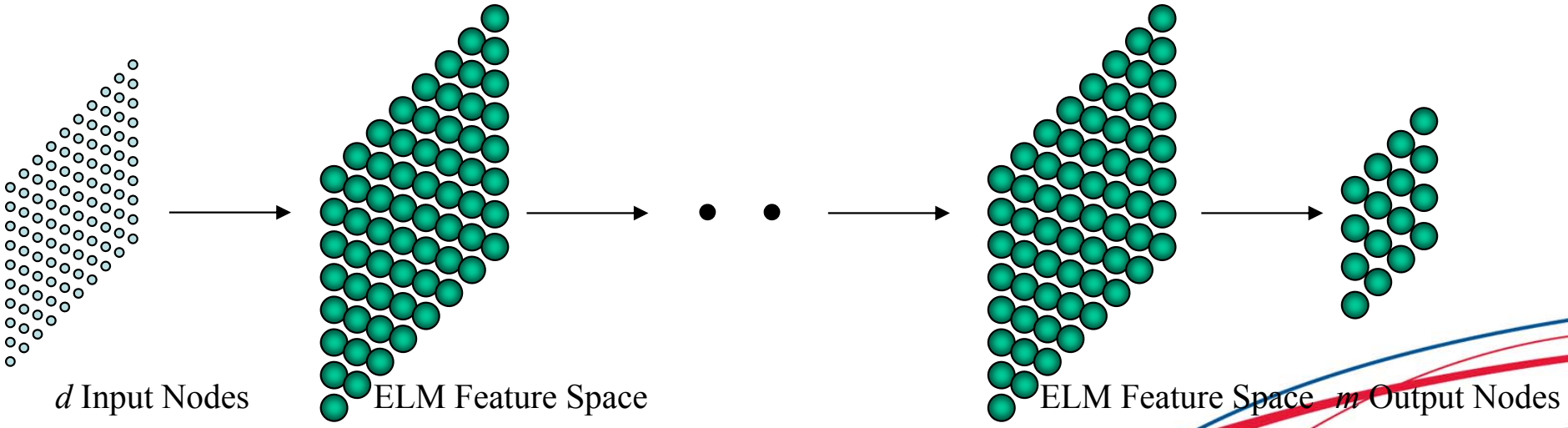
Part II

Hierarchical ELM

- Layer-wise learning
- but learning without iteratively tuning hidden neurons
- output weights analytically calculated by closed-forms solutions in many applications

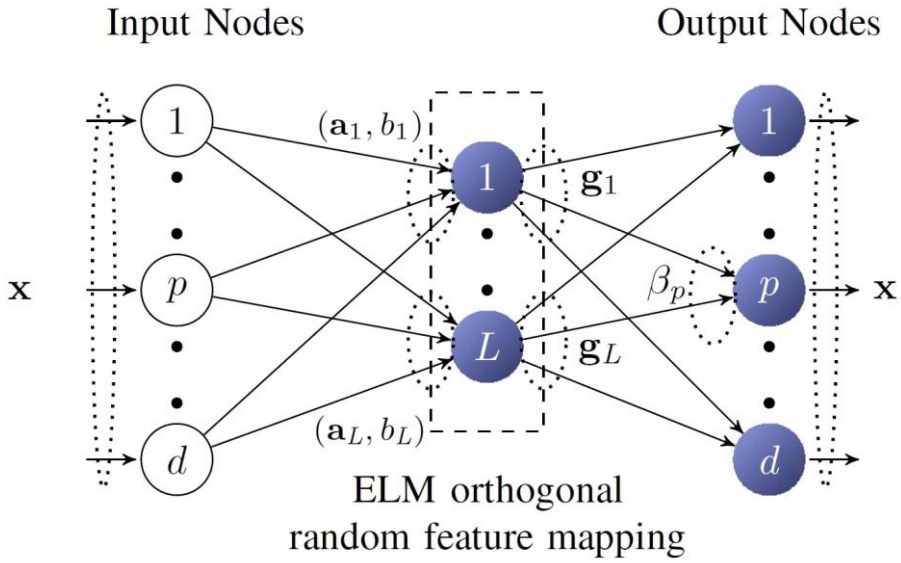
Multi-Layer ELM



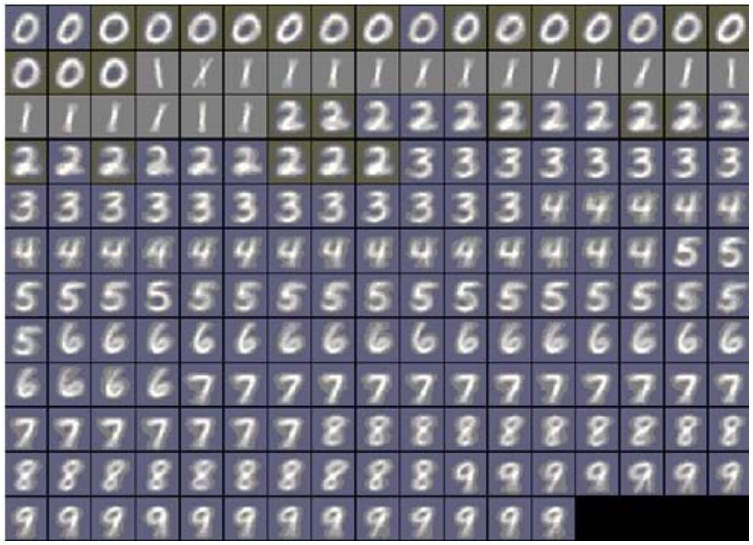


Different from Deep Learning, All the hidden neurons in ELM as a whole are not required to be iteratively tuned

ELM as Auto-Encoder (ELM-AE)

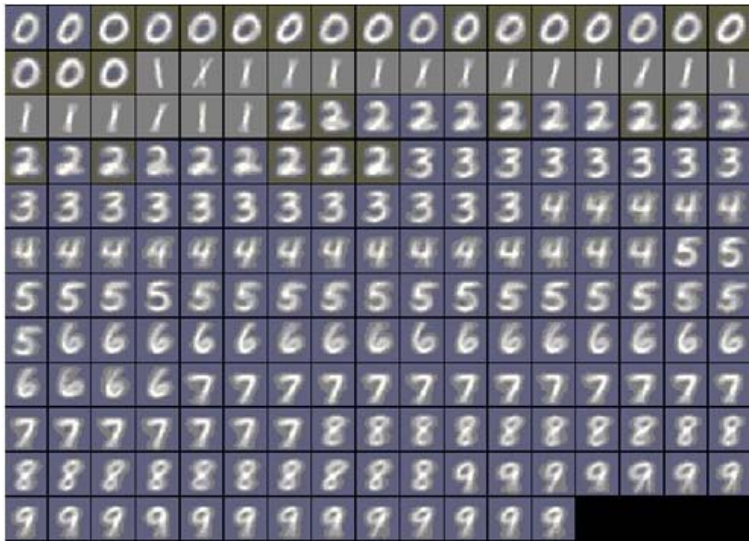


$d > L$: Compressed Representation
$d = L$: Equal Dimension Representation
$d < L$: Sparse Representation

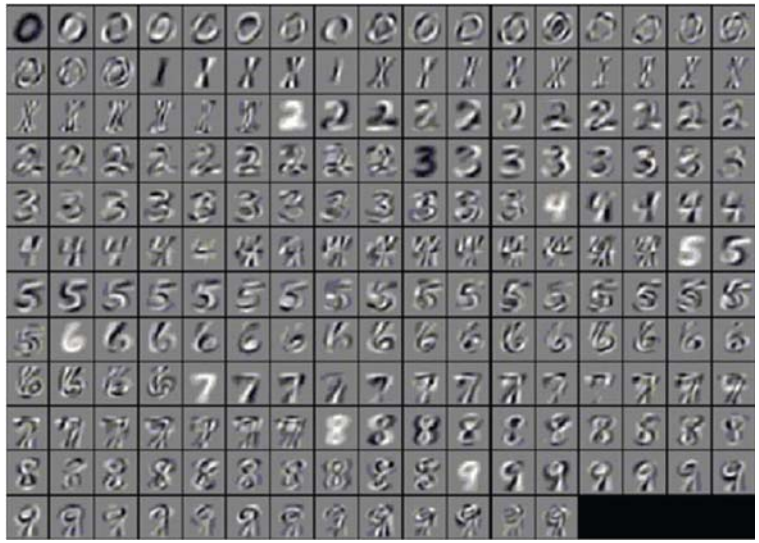


Features represented by the output weights of ELM-AE of MNIST OCR Datasets (with 60000 training samples and 10000 testing samples)

ELM as Auto-Encoder (ELM-AE)



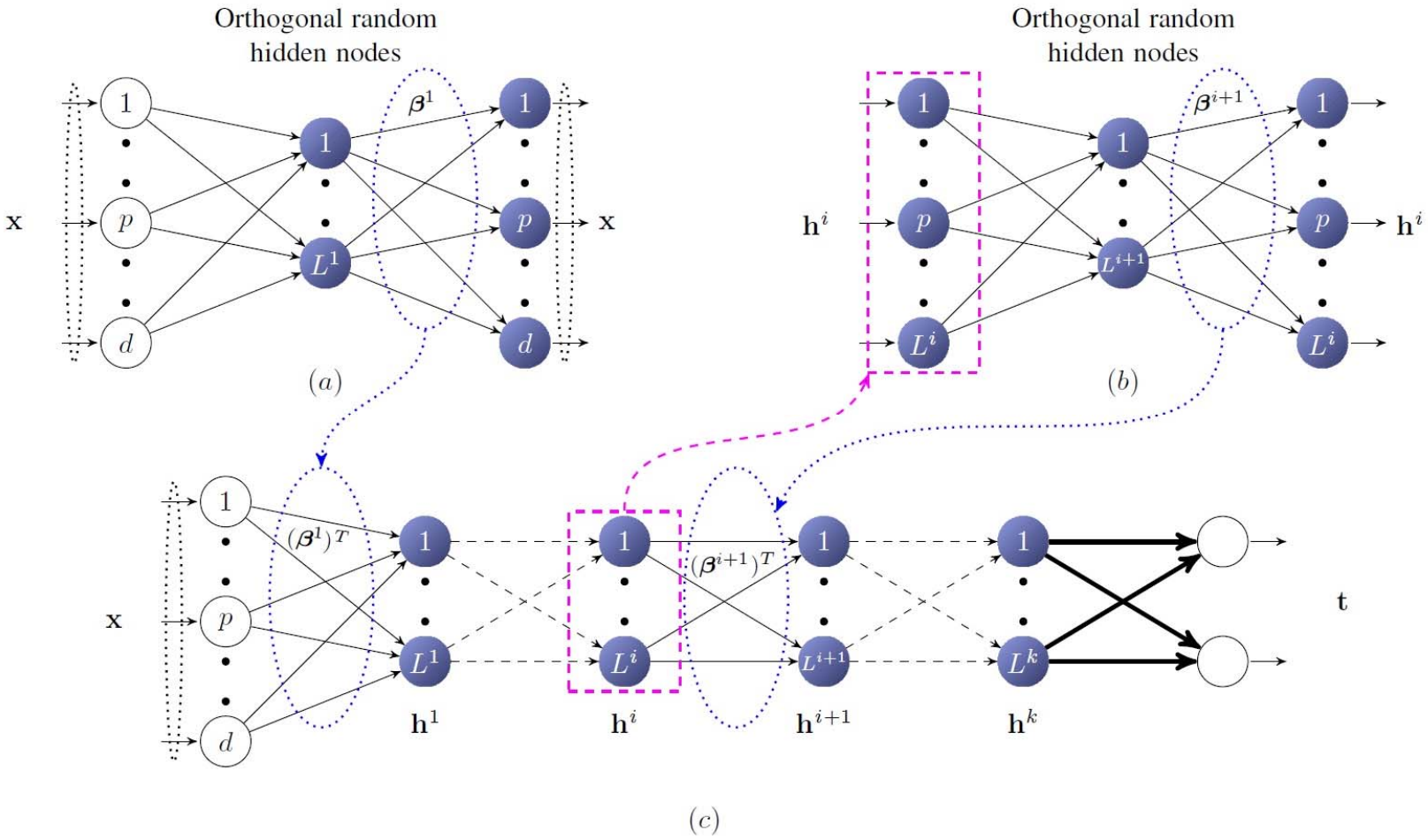
(a) ELM



(b) SVD

ELM-AE vs. singular value decomposition. (a) The output weights $\boldsymbol{\beta}$ of ELM-AE and (b) rank 20 SVD basis shows the feature representation of each number (0–9) in the MNIST dataset.

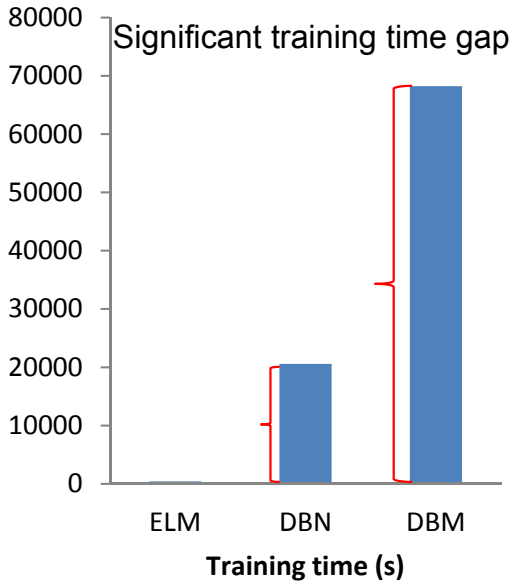
ELM as Auto-Encoder (ELM-AE)



ELM-AE based multi-Layer ELM (ML-ELM): Different from Deep Learning, no iteration is required in tuning the entire multi-layer feedforward networks

ELM vs Deep Learning

Learning Methods	Testing Accuracy	Training Time
H-ELM [Chenwei Deng, et al, 2015]	**99.14**	**281.37s**
Multi-Layer ELM (784-700-700-15000-10) [Huang, et al 2013]	**99.03±0.04**	**444.7s**
Deep Belief Networks (DBN) (748-500-500-2000-10)	98.87	20580s **(5.7 hours)**
Deep Boltzmann Machines (DBM) (784-500-1000-10)	99.05	68246s **(19 hours)**
Stacked Auto Encoders (SAE)	98.6	**> 17 hours**
Stacked Denoising Auto Encoders (SDAE) [Huang, et al 2013]	98.72	**> 17 hours**

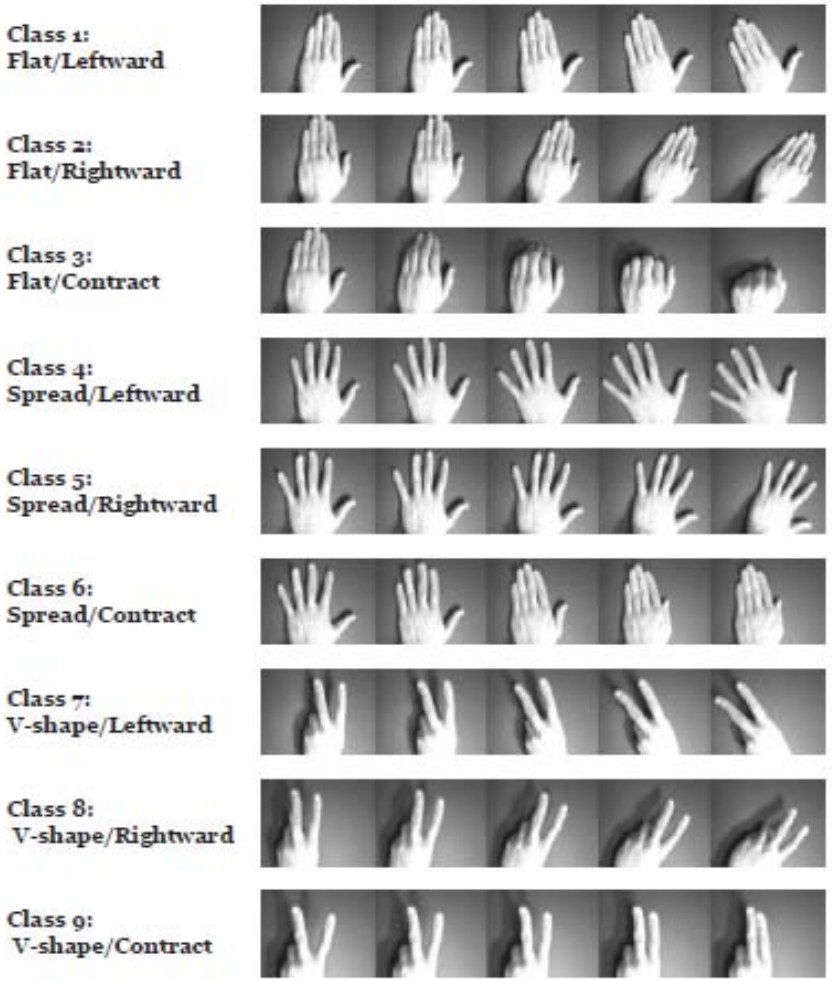


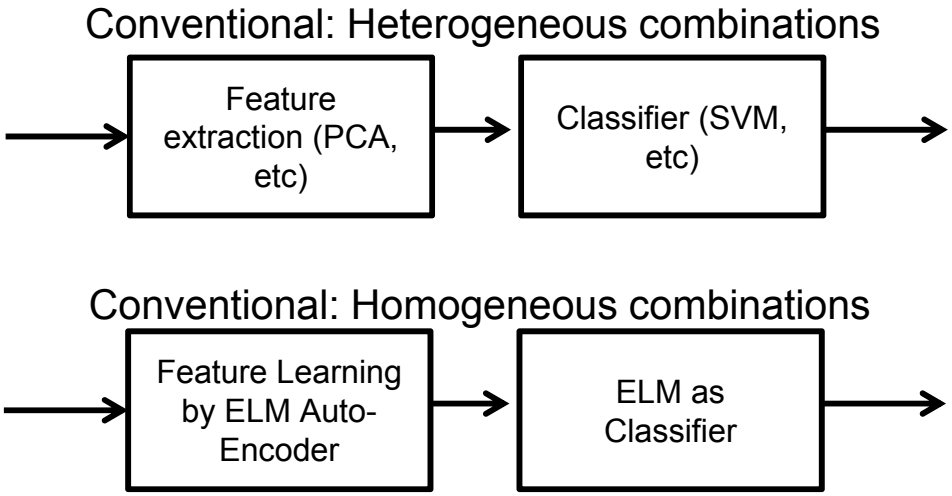
L. L. C. Kasun, et al, "Representational Learning with Extreme Learning Machine for Big Data," *IEEE Intelligent Systems*, vol. 28, no. 6, pp. 31-34, 2013.
J. Tang, et al, "Extreme Learning Machine for Multilayer Perceptron," (in press) *IEEE Transactions on Neural Networks and Learning Systems*, 2015.

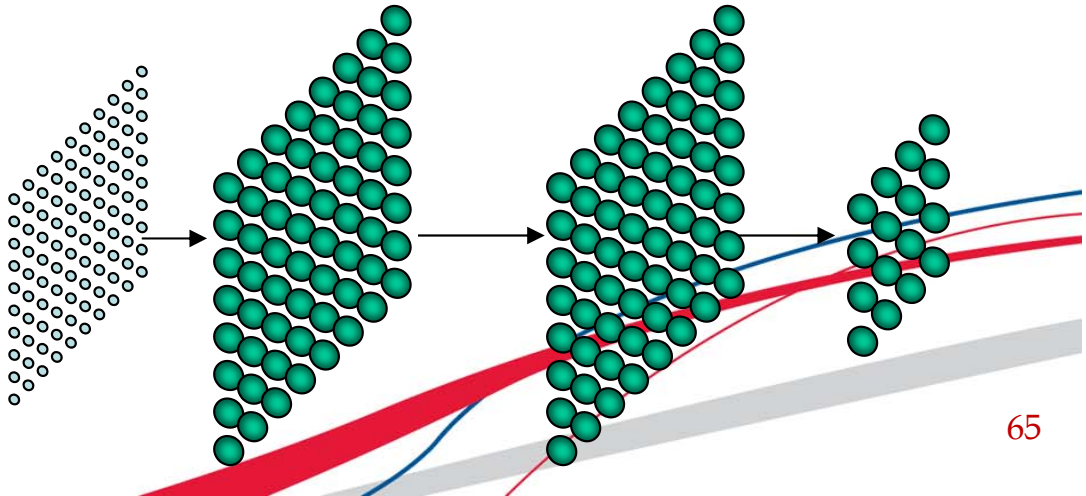
Human Action Recognition

Methods	ELM Based	Tensor canonical correlation	Tangent bundles on special manifolds
Accuracies	99.4	85	93.4

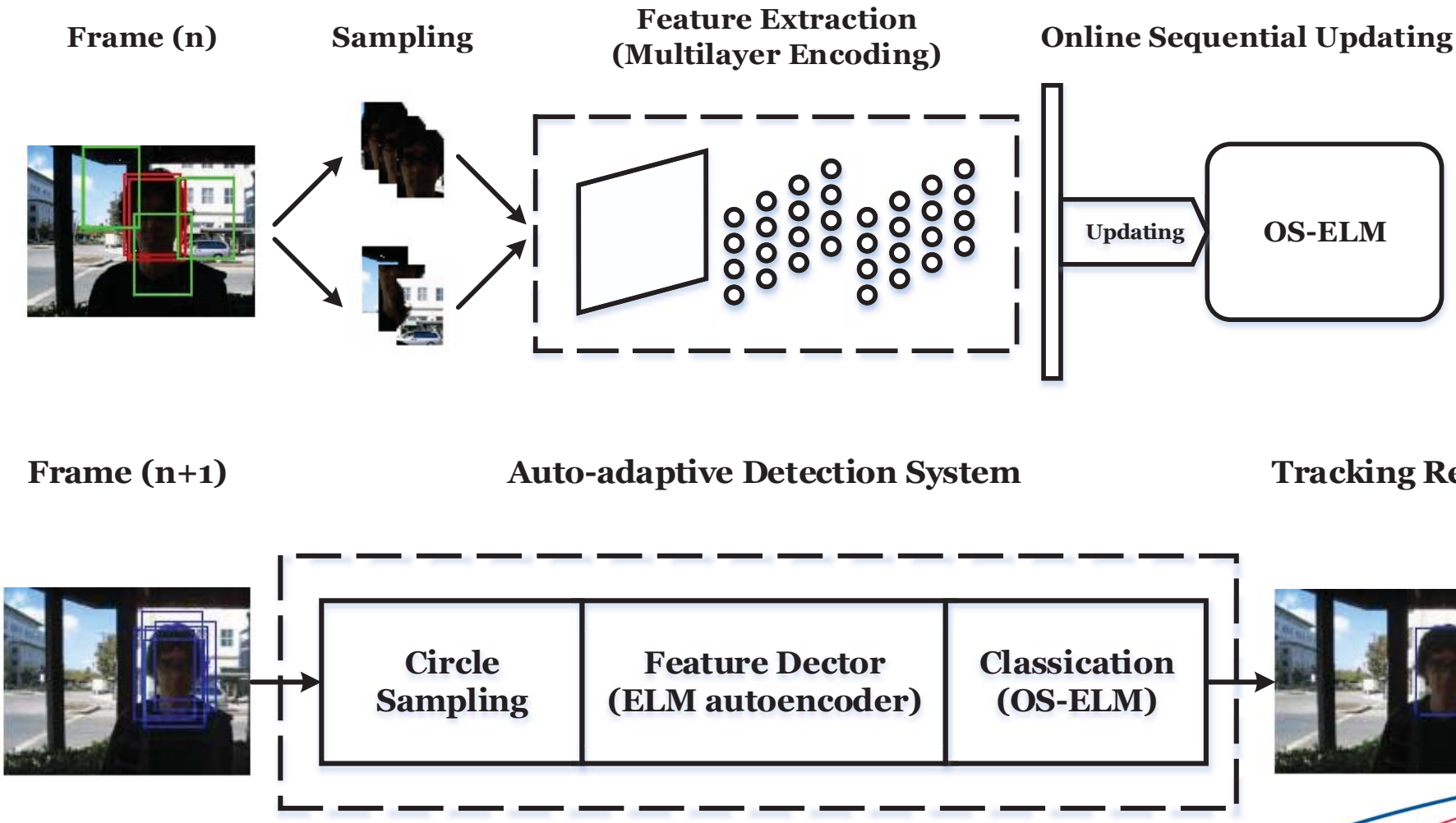
[Deng, et al 2015]







Target Tracking



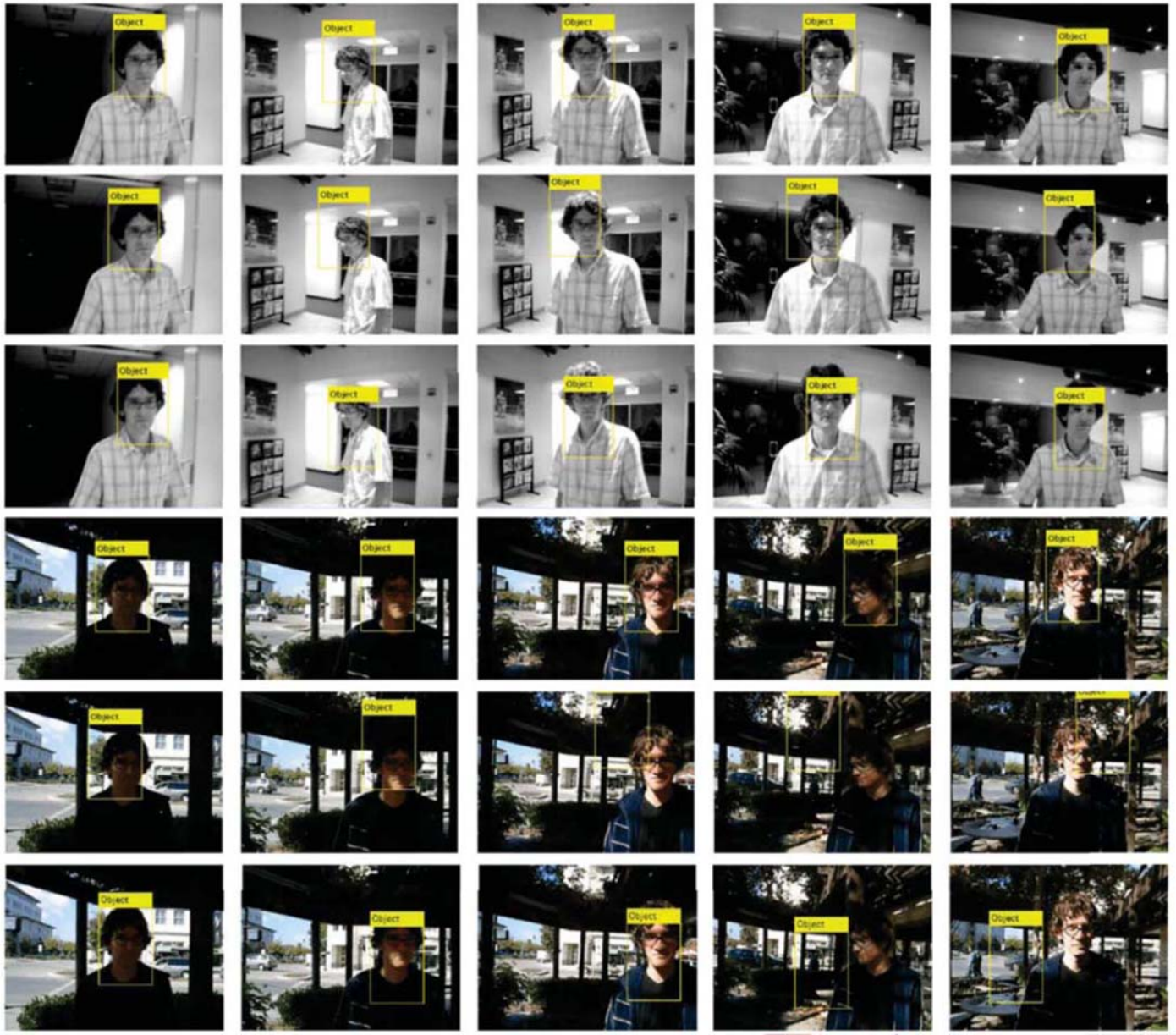
J. Xiong, et al, "Extreme Learning Machine for Multilayer Perceptron", *IEEE Transactions on Neural Networks and Learning Systems*, 2015.

Target Tracking

ELM

Compressive Tracking (CT)

Stacked Autoencoder (SDA)



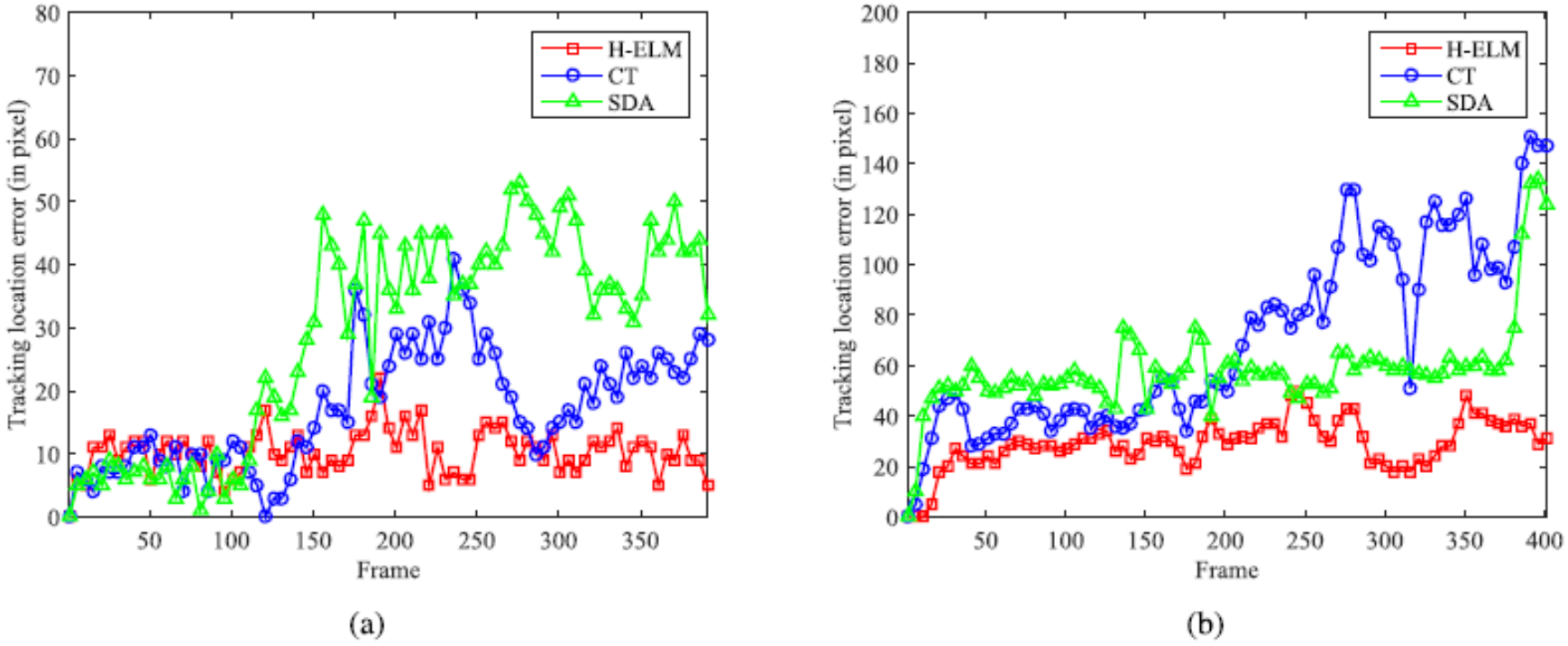
ELM

Compressive Tracking (CT)

Stacked Autoencoder (SDA)

J. Xiong, et al, "Extreme Learning Machine for Multilayer Perceptron", *IEEE Transactions on Neural Networks and Learning Systems*, 2015.

Target Tracking

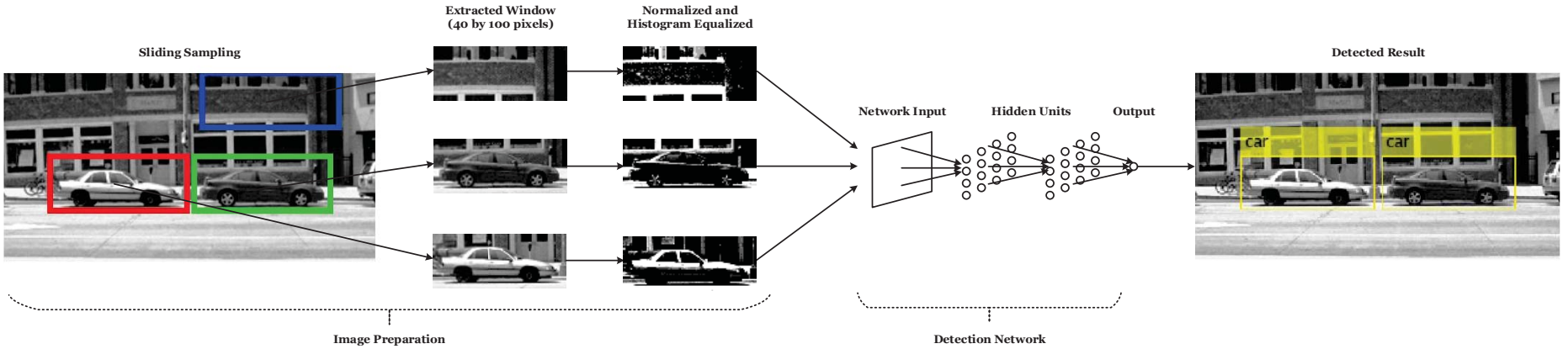


Comparison of tracking location error using H-ELM, CT, and SDA on different data sets. (a) *David Indoor*. (b) *Trellis*

Car Detection

Methods	ELM Based	Contour based learning	SDA
Accuracies	95.5	92.8	93.3
Time	**46.78 s**		3262.30 s

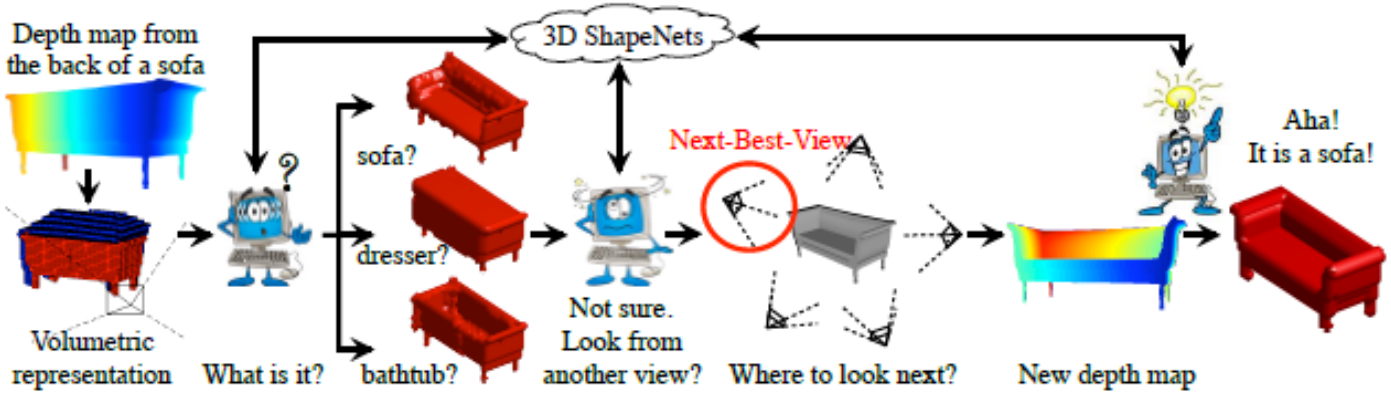
[Deng, et al 2014]



J. Xiong, et al, "Extreme Learning Machine for Multilayer Perceptron", *IEEE Transactions on Neural Networks and Learning Systems*, 2015.

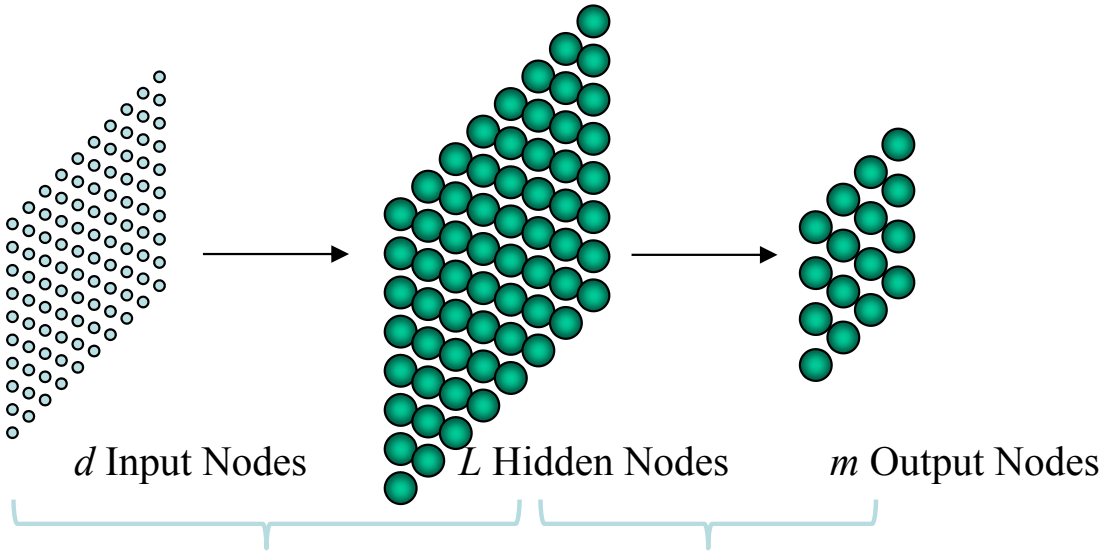
ELM vs Deep Learning

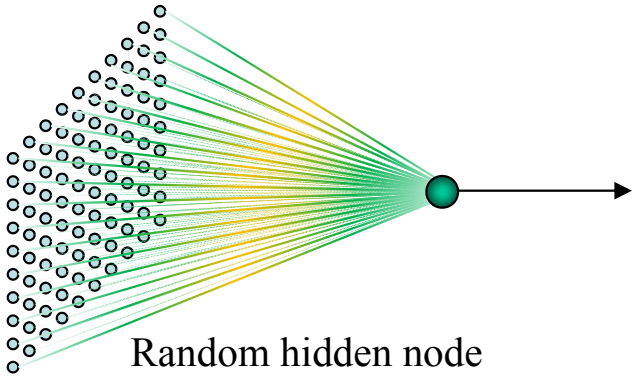
Learning Methods	Testing Accuracy	Training Time
ELM-AE	**86.45**	**602s**
3D ShapeNets (Convolutional Deep Belief Network)	86.5	**Two days**
[Kai XU, Zhige XIE, NUDT, personal communication 2014]		

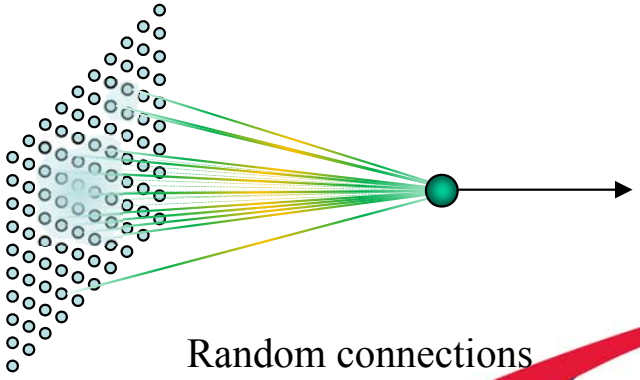


Princeton/MIT/CUHK's 3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction [Wu, et al 2014]

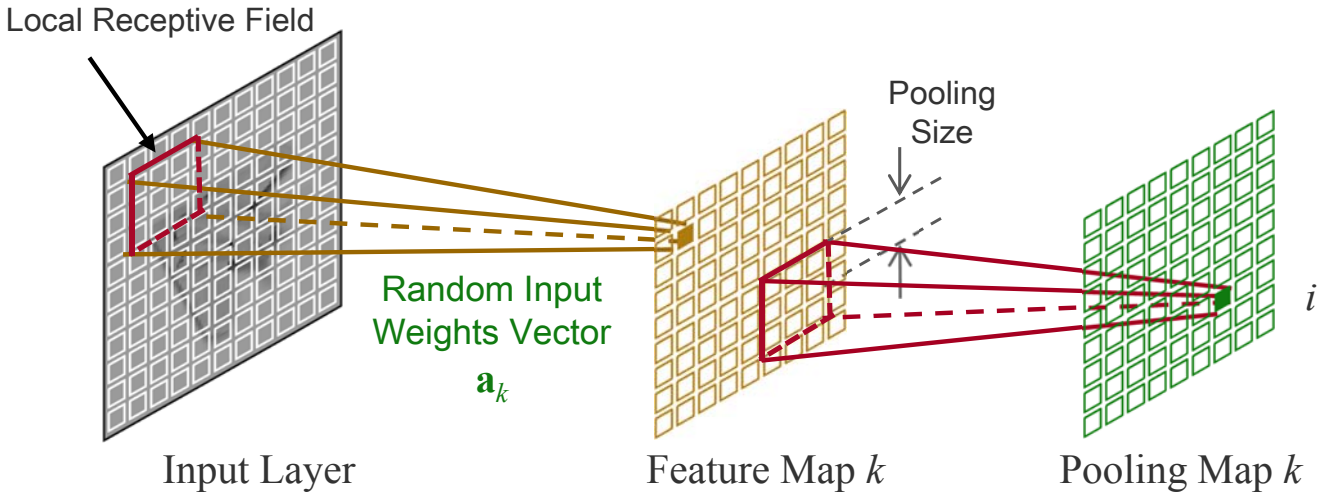
ELM Theory on Local Receptive Fields and Super Nodes

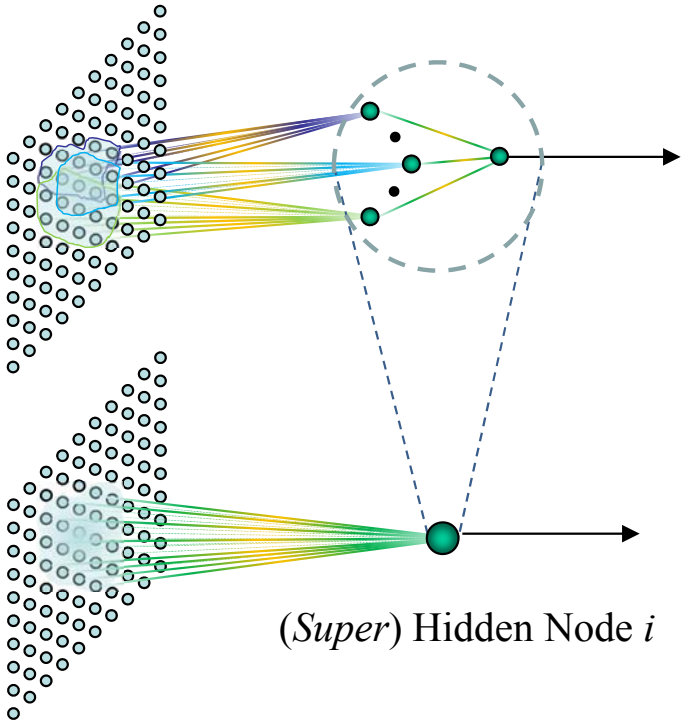






ELM Theory on Local Receptive Fields and Super Nodes





Convolutional nodes and pooling are one of local receptive fields in ELM, but there may have many more.

Similar to sigmoid nodes in feedforward networks, RBF nodes in RBF networks, etc, convolutional nodes in CNN can be considered one type of nonlinear piecewise hidden nodes used in ELM

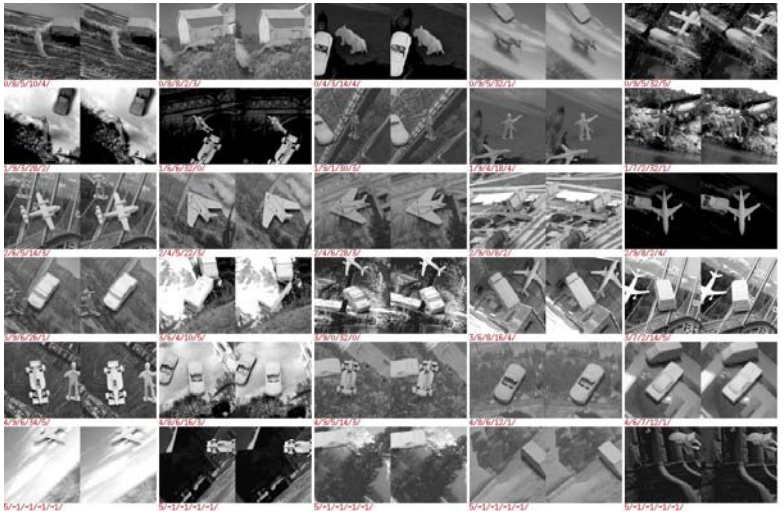
ELM Theory on Local Receptive Fields and Super Nodes

- **Learned from Jürgen Schmidhuber and Dong Yu in INNS BigData, San Francisoc, August 10-12, 2015, and Deep Learning in wiki**
  - First Deep NNs: Ivakhnenko, et al, 1965
  - Basic CNN: Fukushima 1979
  - Back propagation applied to CNNs: LeCun, et al, 1989
    - Mainly on MNIST OCR, but need to spend 3 days
  - Max-Pooling: Weng 1992
- **ELM learning algorithms can also be applied to CNN so that tuning hidden neurons are not required, and meanwhile ELM naturally provides theoretical support to and underpin CNN and Max-Pooling.** **[Huang, et al, 2007, 2015]**

G.-B. Huang and L. Chen, "Convex Incremental Extreme Learning Machine," *Neurocomputing*, vol. 70, pp. 3056-3062, 2007.
G.-B. Huang, et al, "Local Receptive Fields Based Extreme Learning Machine," *IEEE Computational Intelligence Magazine*, vol. 10, no. 2, pp. 18-29, 2015.

ELM vs Deep Learning

Learning Methods	Testing Accuracy
ELM	**97.3%**
Tiled Convolutional Neural Nets	96.1%
Convolutional Neural Nets	94.4%
3D DBNs	93.5%
DBMs	92.8%
SVMs	88.4%



NORB Dataset

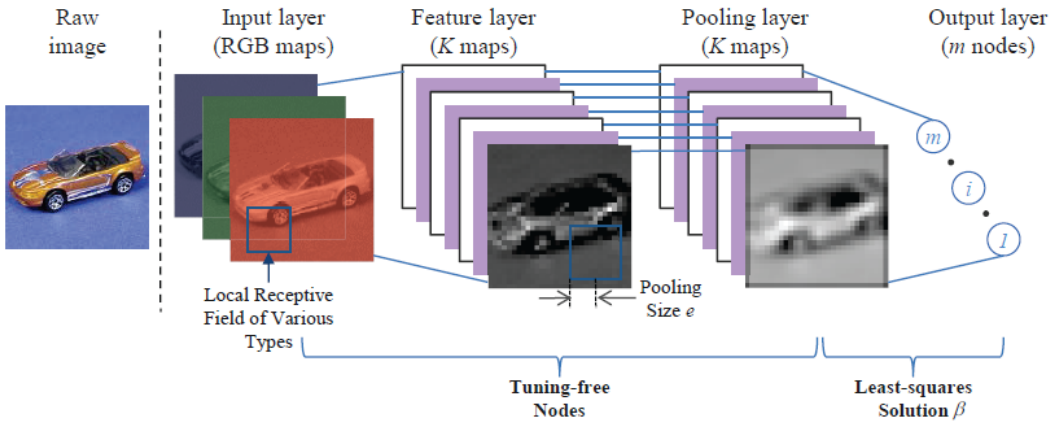
Training time in NORB Data	
DBN	ELM
13 h	**0.1h**

G.-B. Huang, et al, "Local Receptive Fields Based Extreme Learning Machine," *IEEE Computational Intelligence Magazine*, vol. 10, no. 2, pp. 18-29, 2015.

ELM vs Deep Learning

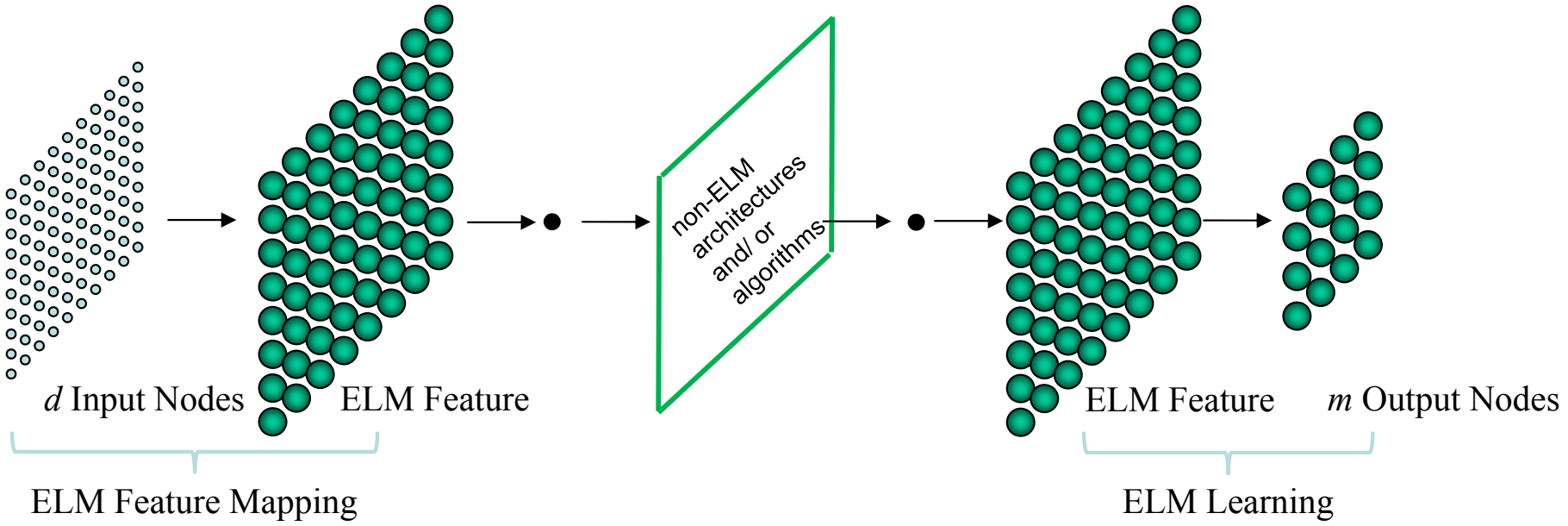
Learning Methods	Testing Error Rate
ELM	**0.02%**
Convolutional Neural Nets (CNN)	28.51%
CNN+video (test images of COIL)	7.75%
CNN++video (COIL-like images)	20.23%

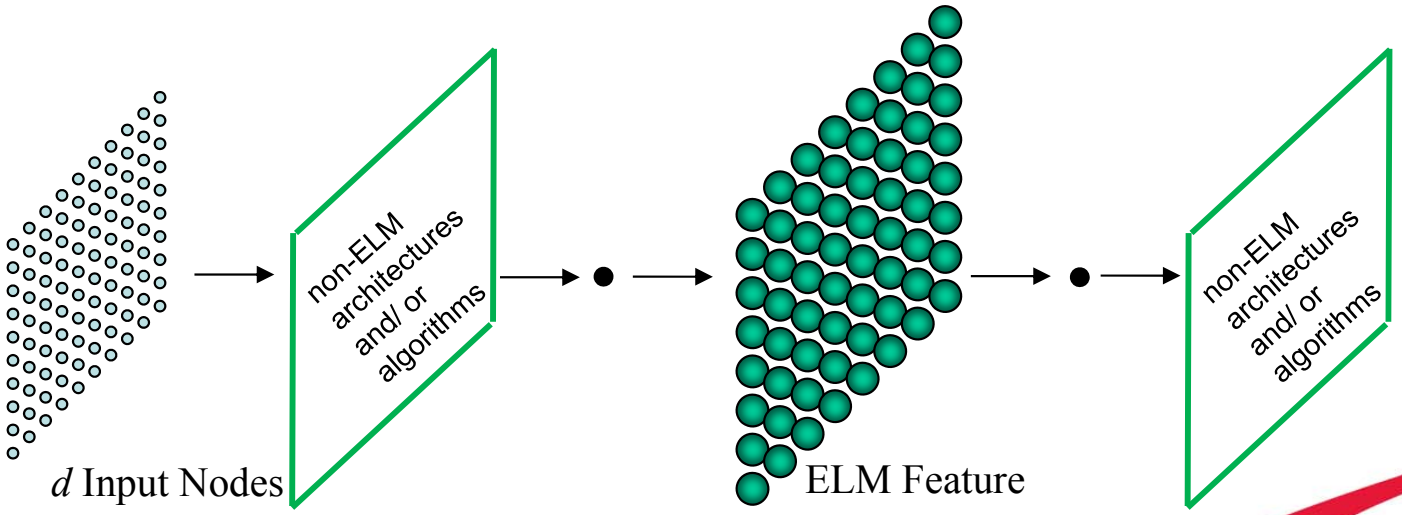
COIL Dataset: 1800 training samples, 5400 testing samples, 100 categories



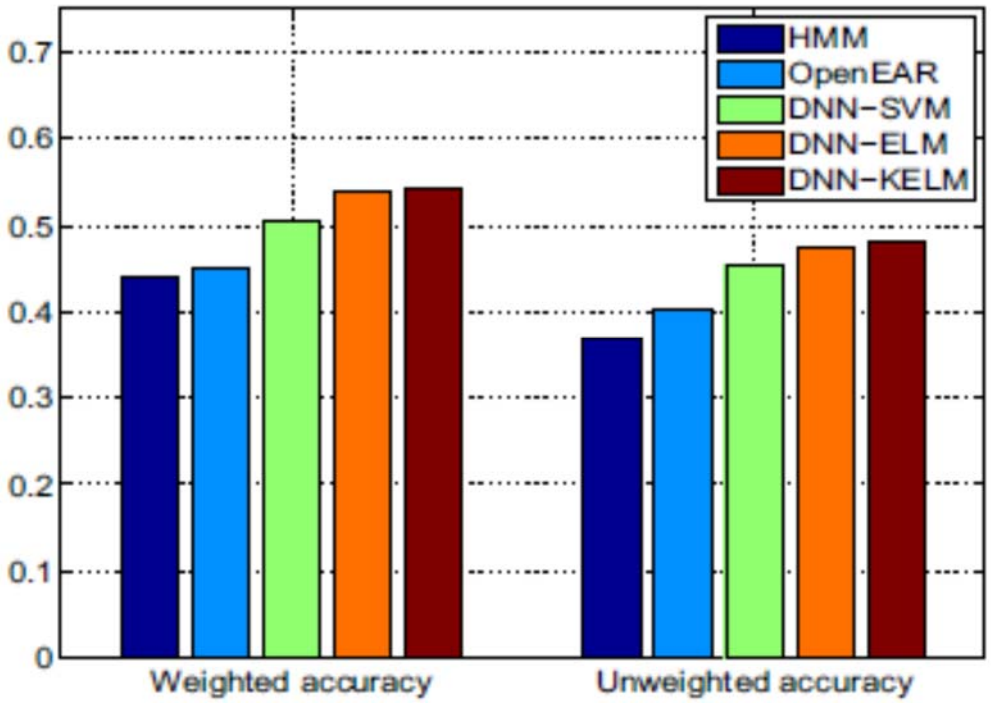
Z. Bai, et al, "Generic Object Recognition with Local Receptive Fields Based Extreme Learning Machine," *2015 INNS Conference on Big Data*, San Francisco, August 8-10, 2015.

ELM Slices


non-ELM architectures and/ or algorithms
d Input Nodes
ELM Feature
ELM Feature
m Output Nodes
ELM Feature Mapping
ELM Learning


non-ELM architectures and/ or algorithms
non-ELM architectures and/ or algorithms
d Input Nodes
ELM Feature

Speech Emotion Recognition (DNN + ELM)

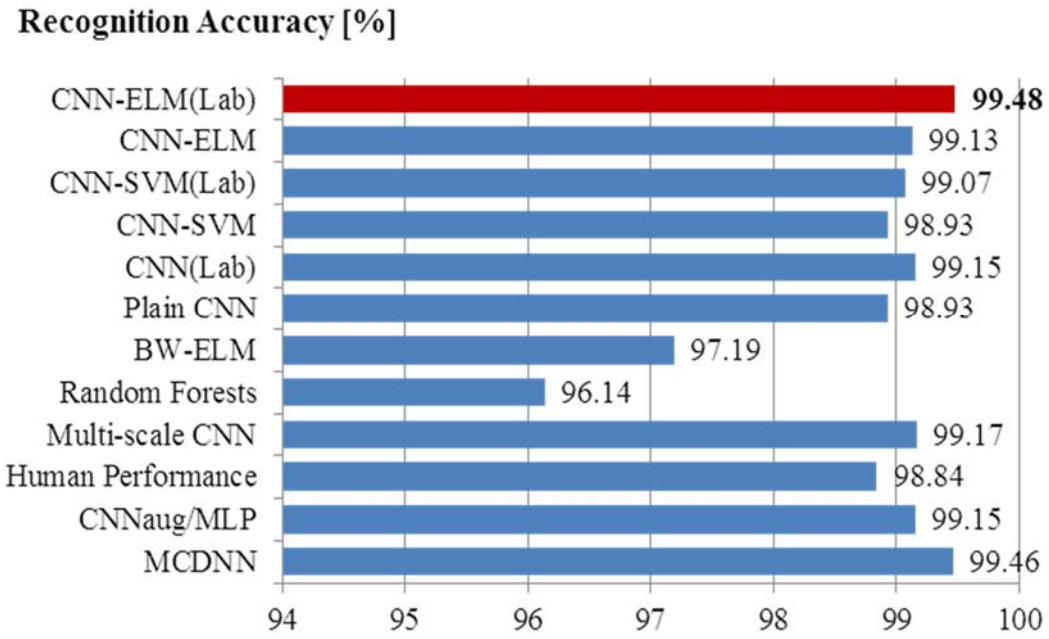


Microsoft Research and Ohio State University [Han, et al 2014]

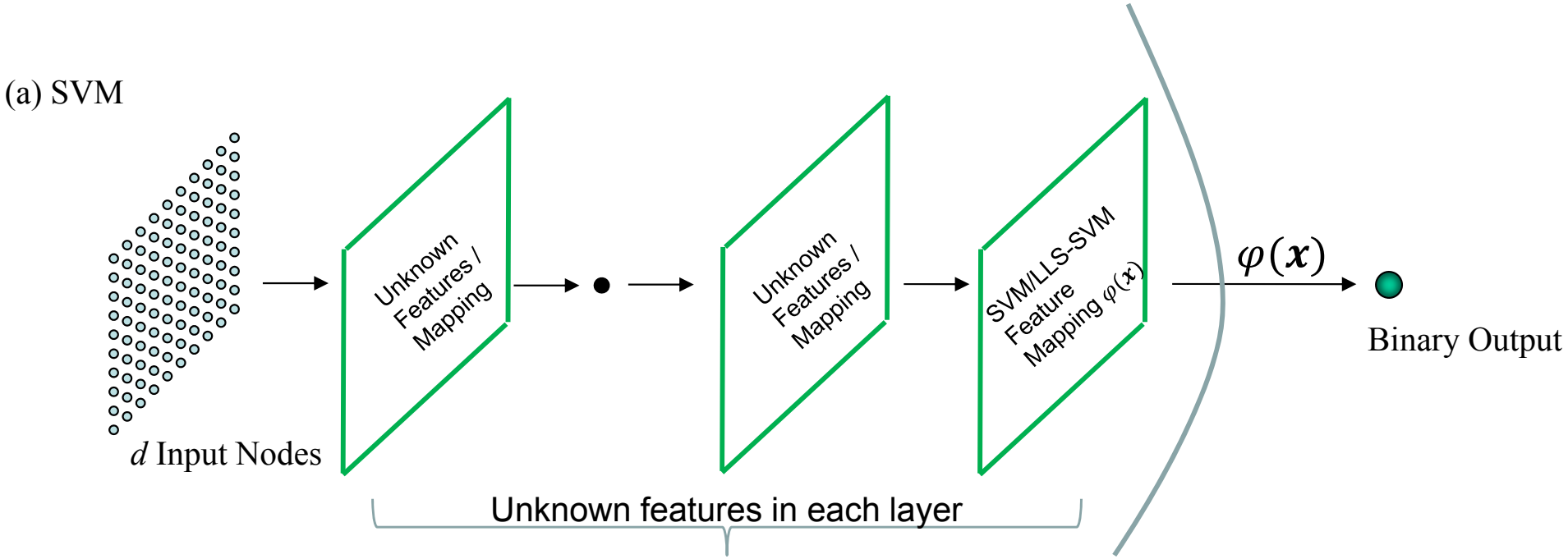
Traffic Sign Recognition (DNN + ELM)

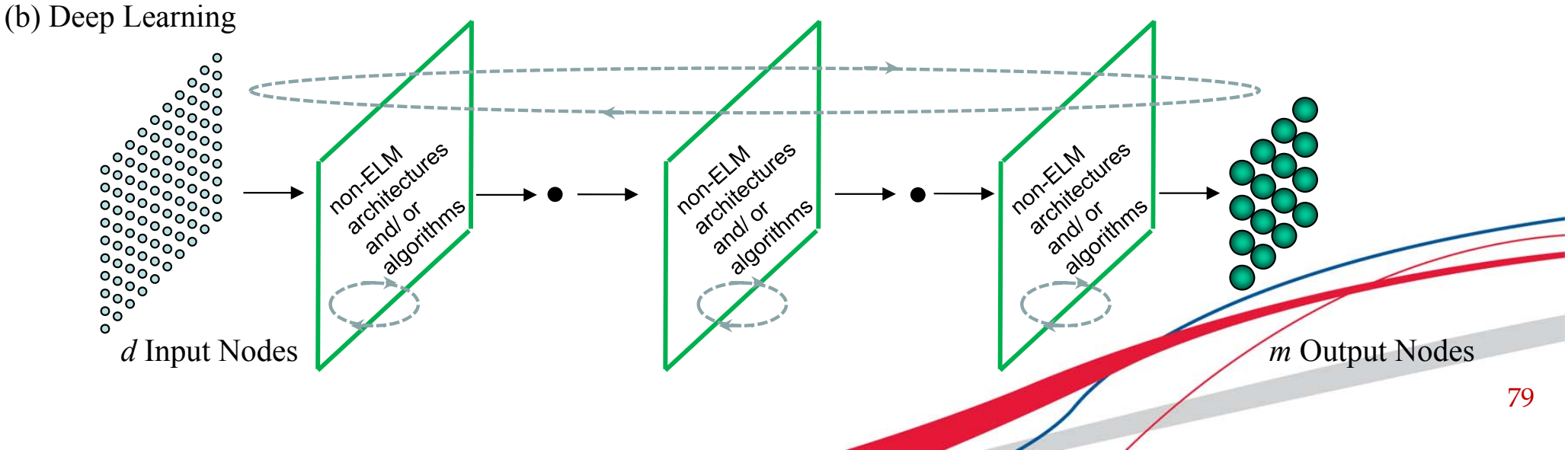
Methods	CNN + ELM Based	MCDNN
Accuracies	99.48%	99.46%
Training time	**5 hours (regular PC)**	37 hours (GPU Implementation)

(ELM may just spend several minutes on training in order to reach 98+% accuracy) [Xu, et al 2015]



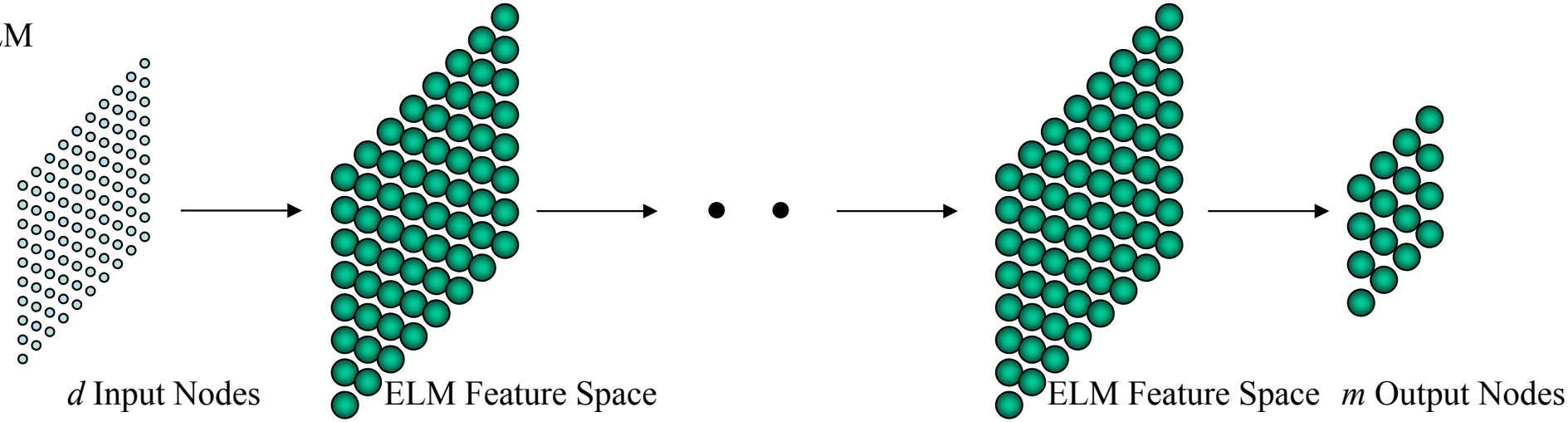
ELM, SVM and Deep Learning





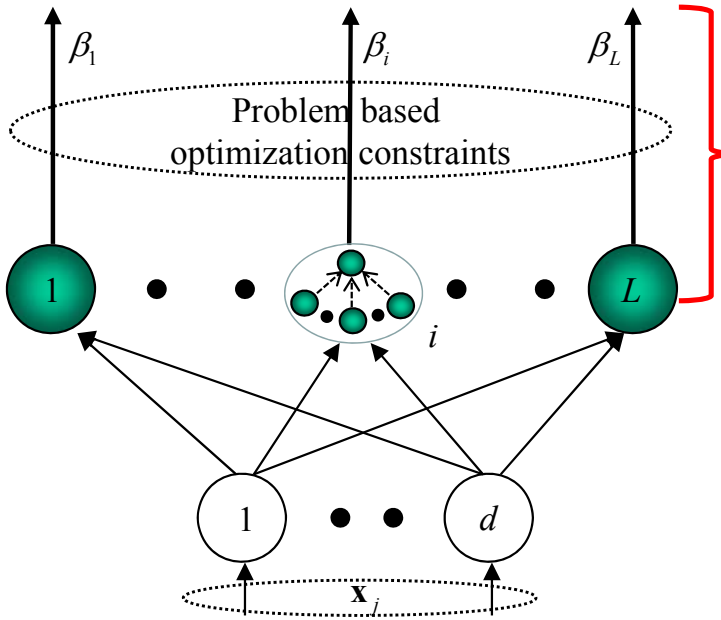
ELM, SVM and Deep Learning

(a) ELM



Different from Deep Learning, All the hidden neurons in ELM as a whole are not required to be iteratively tuned

(b) ELM subnetwork



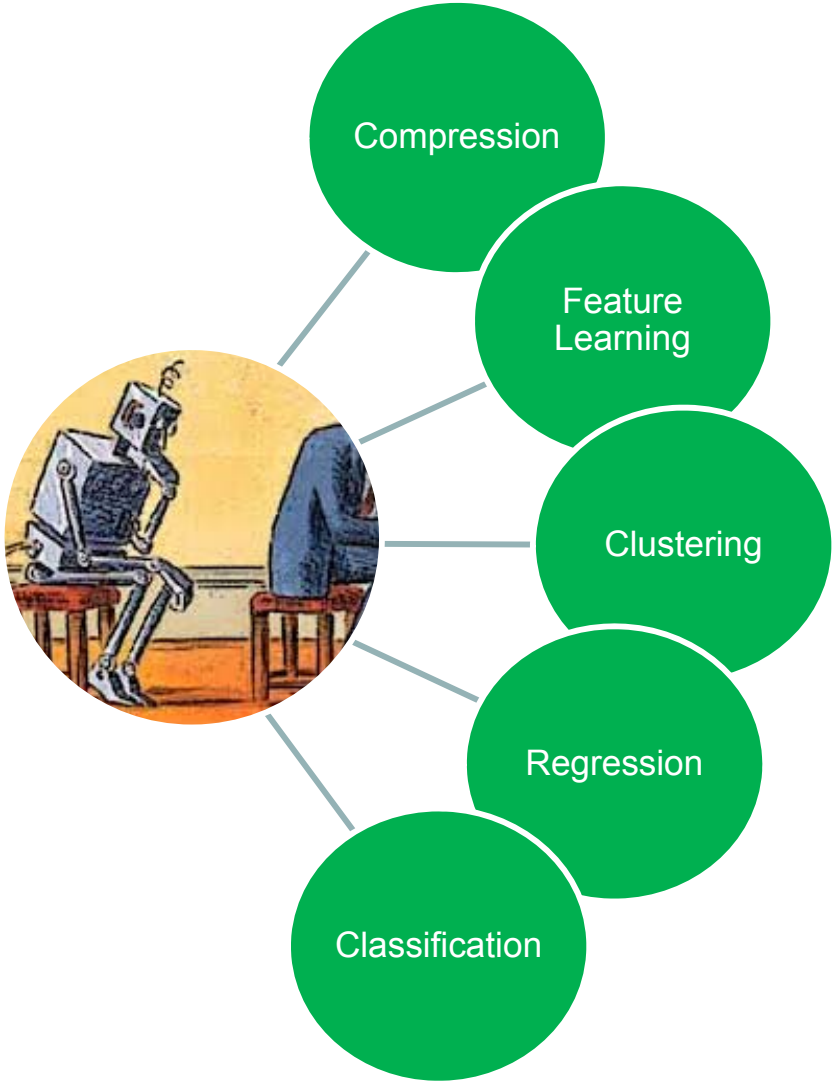
Feature learning
Clustering
Regression
Classification

Hidden nodes need not be tuned. A hidden node can be a subnetwork of several nodes.

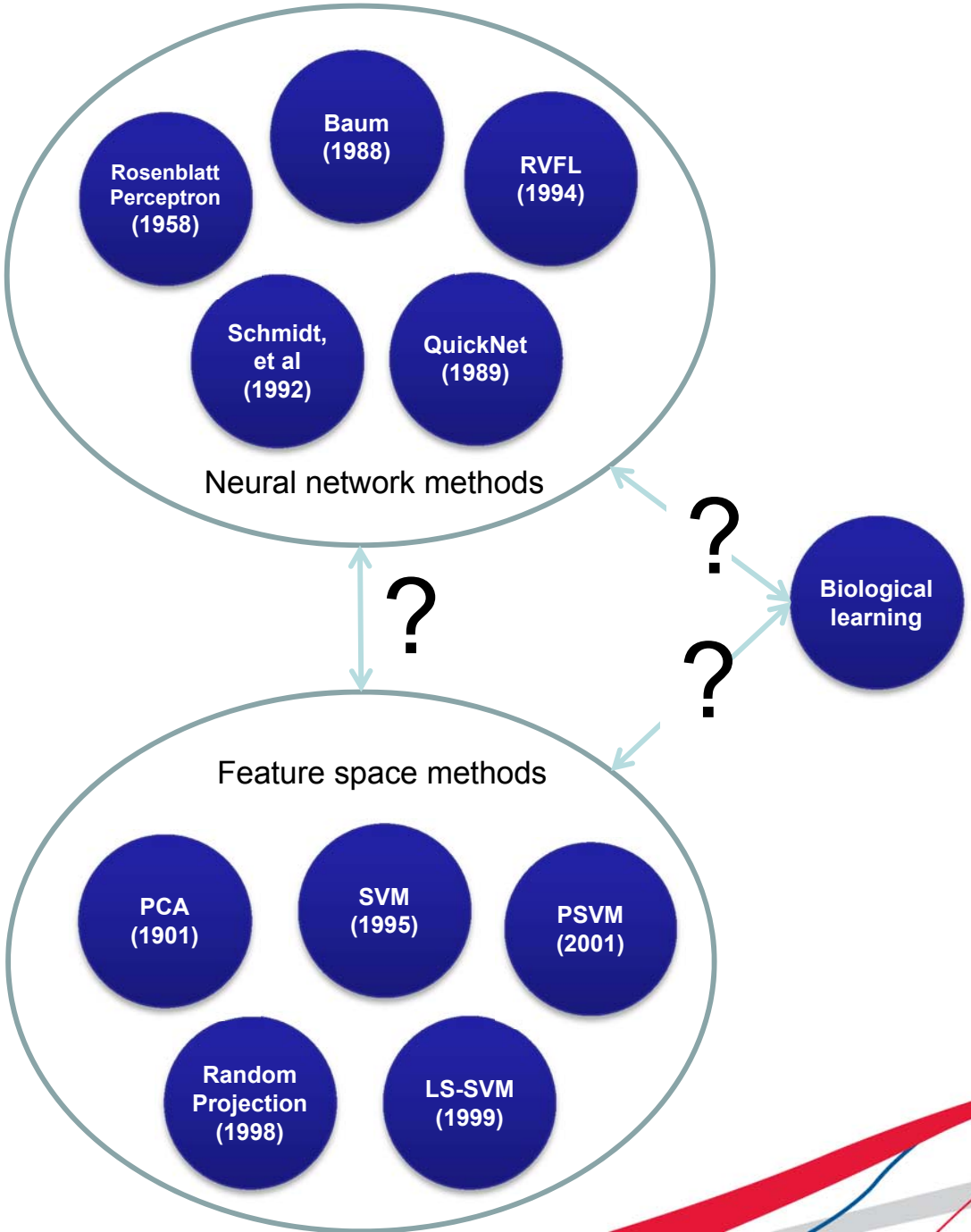
ELM and Deep Learning

Deep Learning	ELMs
Very sensitive to network size, "painful" manually tuning	Stable in a wide range of network size, almost free of human intervention
Difficult in parallel and hardware implementation	Easy in parallel and hardware implementation
Lack of theoretical proof	Rigorously proved in theory
Different models for feature learning, clustering, and classifications	Homogenous models for compression, feature learning, clustering, regression and classification
Impossible for micro level real-time learning and control; huge training time is required; difficult for multi-channel data fusion and decision synchronization	Easy for micro level real-time learning and control, up to thousands times faster, efficient for multi-channel data fusion and potential for decision synchronization
Difficult for online incremental learning and prediction (stream data learning)	Easy for online incremental learning and prediction (streaming data learning)
Only reaching higher accuracy when data is large enough	Reaching higher accuracy in full spectrum of applications, from sparse/small data to large size of applications
Impossible to have hardware designed for universal development	"Brains (devised by ELM)" can be generated before applications are present
Huge computing resources required (GPU required, up to tens of thousands of cores required)	Usually implemented in regular PCs / Laptops / FPGA / Chip; The training time would significantly be reduced if multi cores are used.

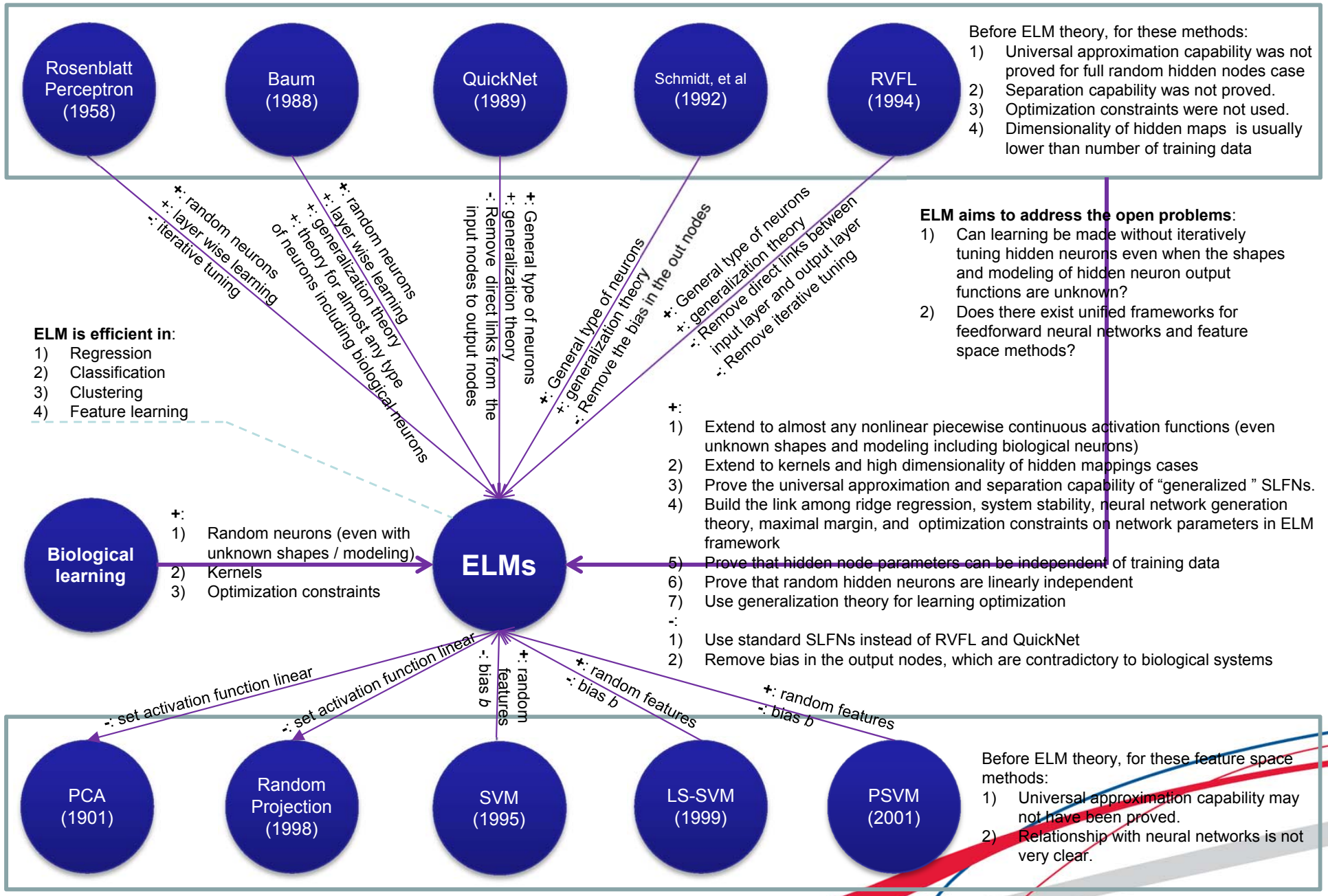
ELM as Fundamentals of Cognition and Reasoning



ELM Filling Gaps …



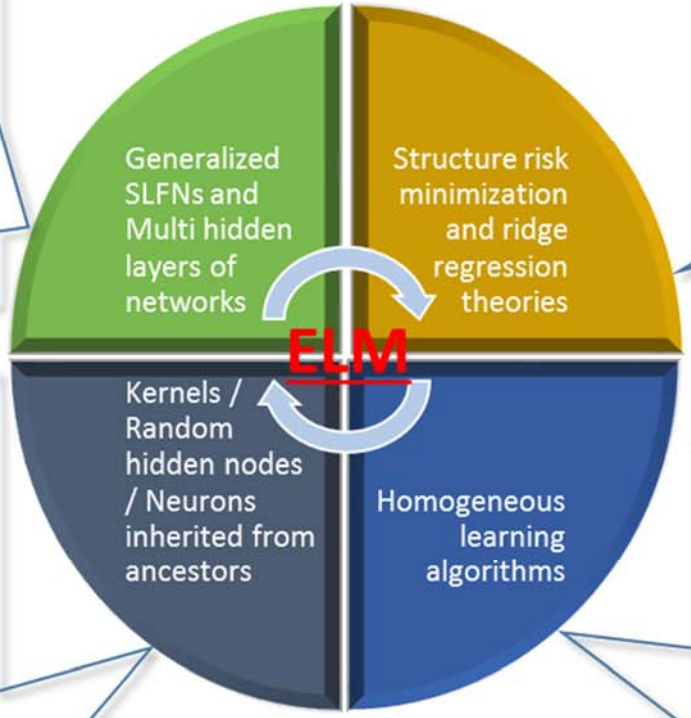
ELM Filling Gaps …



ELM Filling Gaps …

- PCA, random projection, SVM, QuickNet, RVFL, Schmidt *et al.* 1992 are considered isolated before ELM
- ELMs provide unifying network architectures with neither bias in the output nodes nor direct links between the input and output layers
- A hidden node in ELM can be a network
- ELM is a generalized SLFN or multi layers of networks

- Neural network generalization performance theory, stricture risk minimization and ridge regression become consistent in ELM
- Ridge regression theories applied in feature learning, clustering, regression, binary and multi-class applications



- Random hidden nodes / neurons independent of training samples
- Kernels can also be used
- Hidden neurons can be inherited from their ancestors or other systems and thus, tuning is not required
- Wide types of hidden nodes / neurons (in real and complex domains, with known and unknown shapes)
- Universal approximation and classification capabilities proved in theory

- Same network architectures for compression, feature learning, clustering, regression and classification
- Similar learning algorithms for different learning tasks
- Scalable for hierarchical blocks of ELMs

Towards Biological Learning, Cognition and Reasoning?

Biological Learning	ELMs
Stable in a wide range (tens to thousands of neurons in each module)	Stable in a wide range (tens to thousands of neurons in each module)
Parallel implementation	Easy in parallel implementation
"Biological" implementation	Much easier in hardware implementation
Free of user specified parameters	Least human intervention
One module possibly for several types of applications	One network type for different applications
Fast in micro learning point	Fast in micro learning point
Nature in online sequential learning	Easy in online sequential learning
Fast speed and high accuracy	Fast speed and high accuracy
Brains are built before applications	"Brains (devised by ELM)" can be generated before applications are present

Biological Learning vs Computers

- **J. von Neumann, Father of Computers' Puzzles** [Neumann 1951, 1956]
  - Why ``an imperfect (*biological*) neural network, containing many random connections, can be made to perform reliably those functions which might be represented by idealized wiring diagrams" [Rosenblatt 1958]

- **60 Years Later …**
- **Answered by ELM Learning Theory** [Huang, et al 2006, 2007, 2008]
  - "As long as the output functions of hidden neurons are nonlinear piecewise continuous and even if their shapes and modeling are unknown, (*biological*) neural networks with random hidden neurons attain both universal approximation and classification capabilities, and the changes in *finite* number of hidden neurons and their related connections do not affect the overall performance of the networks." [Huang 2014]

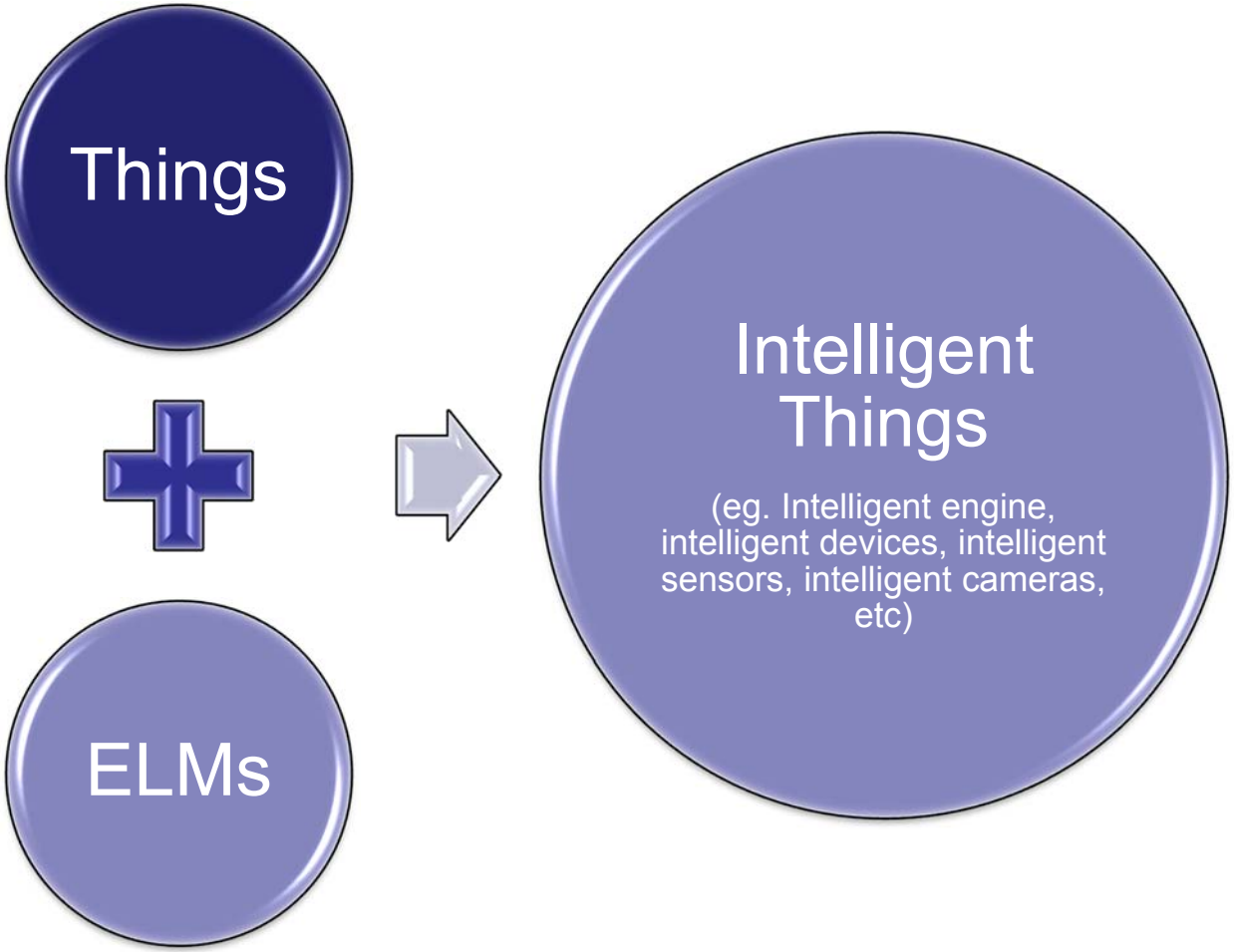
Biological Learning vs Computers

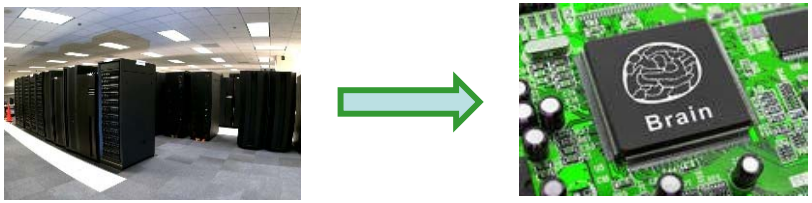
- **ELM Learning Theory** [Huang, et al 2006, 2007, 2008, 2014, 2015]
  - ELM can be used to train wide type of multi hidden layer of feedforward networks:
    - Each hidden layer can be trained by one single ELM based on its role as feature learning, clustering, regression or classification.
    - Entire network as a whole can be considered as a single ELM in which hidden neurons need not be tuned.
  - ELM slice can be ``inserted" into many local parts of a multi hidden layer feedforward network, or work together with other learning architectures / models.
  - A hidden node in an ELM slice (a ``generalized" SLFN) can be a network of several nodes, thus local receptive fields can be formed.
  - In each hidden layer, input layers to hidden nodes can be fully or partially randomly connected according to different continuous probability distribution function.

Biological Learning vs Computers

- **ELM Learning Theory** [Huang, et al 2006, 2007, 2008, 2014, 2015]
  - From ELM theories point of view, the entire multi layers of networks are structured and ordered, but they may be seemingly ``messy" and ``unstructured" in a particular layer or neuron slice. *``Hard wiring" can be randomly built locally with full connection or partial connections.*
  - Co-existence of globally structured architectures and locally random hidden neurons happen to have fundamental learning capabilities of compression, feature learning, clustering, regression and classification.
  - Biological learning mechanisms are sophisticated, we believe that ``learning without tuning hidden neurons" is one of fundamental biological learning mechanisms in many modules of learning systems. Furthermore, *random hidden neurons and ``random wiring"* are only two specific implementations of such ``learning without tuning hidden neurons" learning mechanisms.

Internet of Intelligent Things





Society of Intelligent Things

Three Stages of Intelligent Things

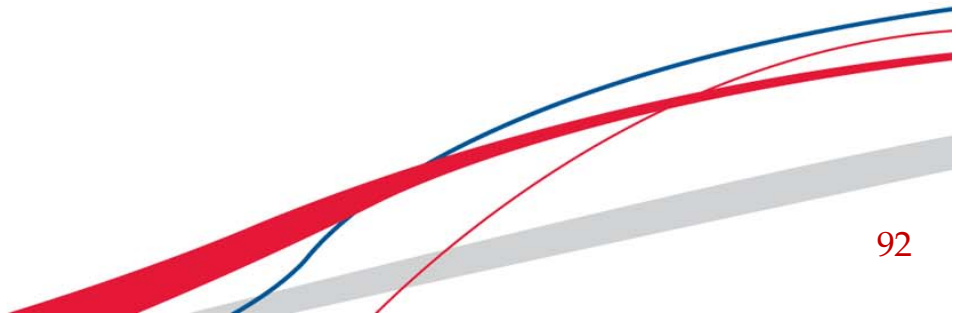
Internet of Things

Internet of Intelligent Things

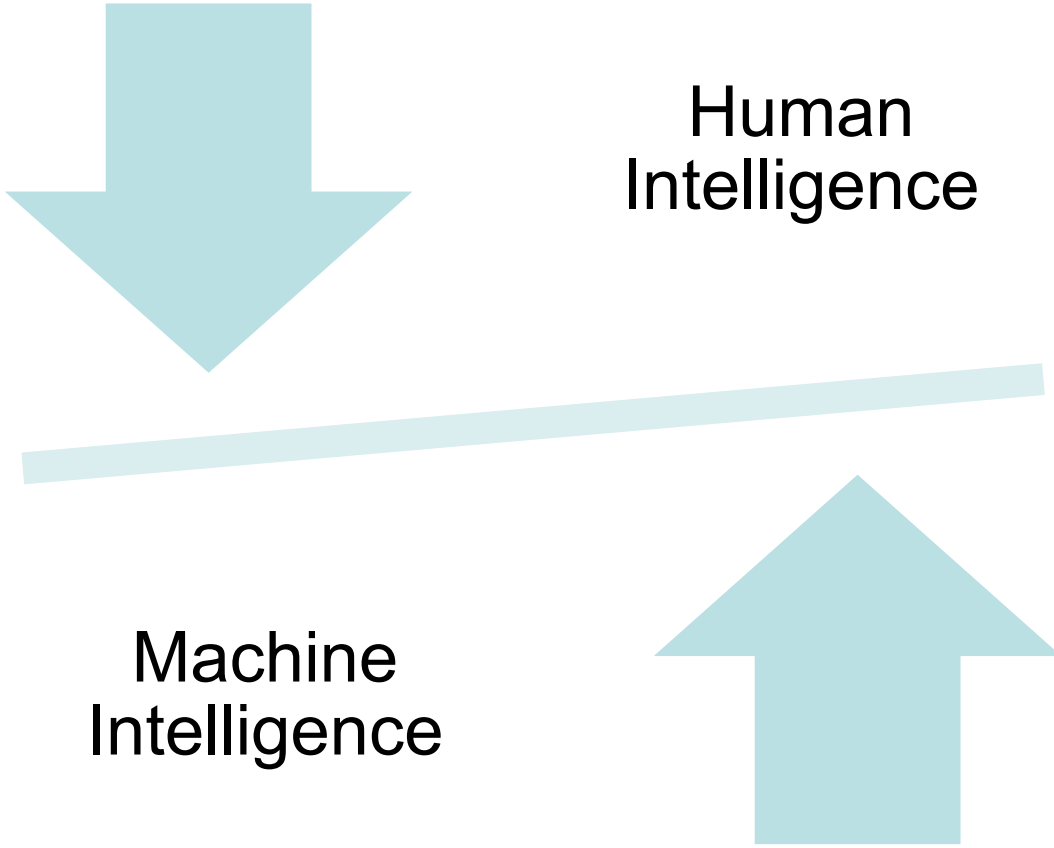
- Intelligent things with ELMs
- Smart materials, smart sensors

Society of Intelligent Things

- Internet disappearing?
- From living thing intelligence to machine intelligence?



Human Intelligence vs Machine Intelligence



Part III: ELM Theories and Incremental/Sequential ELM

Outline

Outline

Outline

Outline

Conventional Approximation Capability Theory

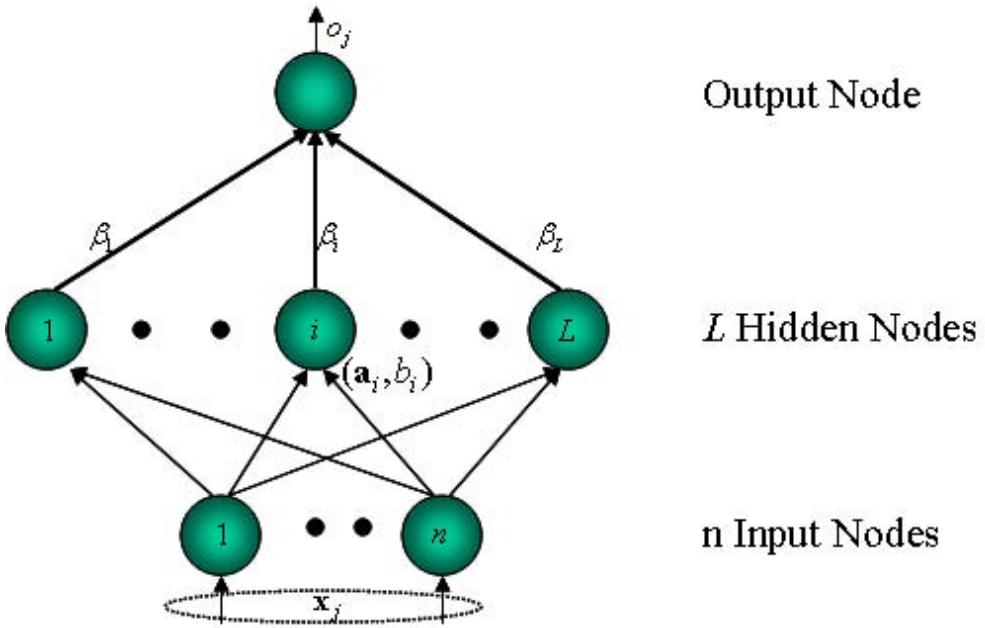


Figure 1: Feedforward Network Architecture.

Conventional Existence Theorem

1. Any continuous target function $f(\mathbf{x})$ can be approximated by SLFNs with some kind of hidden nodes and with appropriate values for learning parameters (hidden node parameters $(\mathbf{a}_i, b_i)$) and output weights β_i.
2. In other words, given any small positive value ϵ, for sigmoid type or RBF type of SLFNs, there exist a set of hidden node parameters $(\mathbf{a}_i, b_i)$ and appropriate number (L) of hidden nodes such that

$$\|f_L(\mathbf{x}) - f(\mathbf{x})\| < \epsilon \qquad (1)$$

M. Leshno, et al., "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861-867, 1993.

J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, pp. 246-257, 1991.

ELM Learning Theory

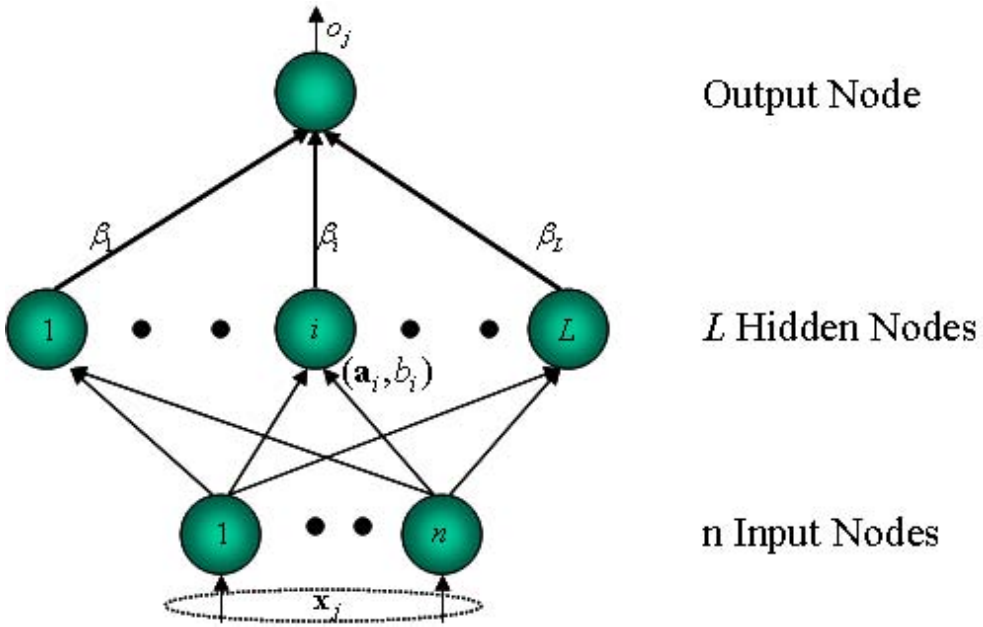


Figure 2: Feedforward Network Architecture: any type of nonlinear piecewise continuous $G(\mathbf{a}_i, b_i, \mathbf{x})$.

New Learning Theory

Given a SLFN with any nonconstant piecewise continuous hidden nodes $G(\mathbf{x}, \mathbf{a}, b)$, if $span\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{C}^d \times C\}$ is dense in L^2, for any given positive value ϵ, for any continuous target function f and any randomly generated sequence $\{(\mathbf{a}_n, b_n)_{n=1}^{L}\}$, there exists an integer $L_0 > 0$ such that when $L > L_0$

$$\left\| f(\mathbf{x}) - \sum_{n=1}^{L} \beta_n g_n \right\| < \epsilon \qquad (2)$$

holds with probability one if $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$, $g_n = G(\mathbf{a}_n, b_n, \mathbf{x})$, $i = 1, \cdots, L$.

G.-B. Huang, et al., "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879-892, 2006.

G.-B. Huang, et al., "Convex incremental learning machine," *Neurocomputing*, vol. 70, pp. 3056-3062, 2007.

ELM Learning Theory

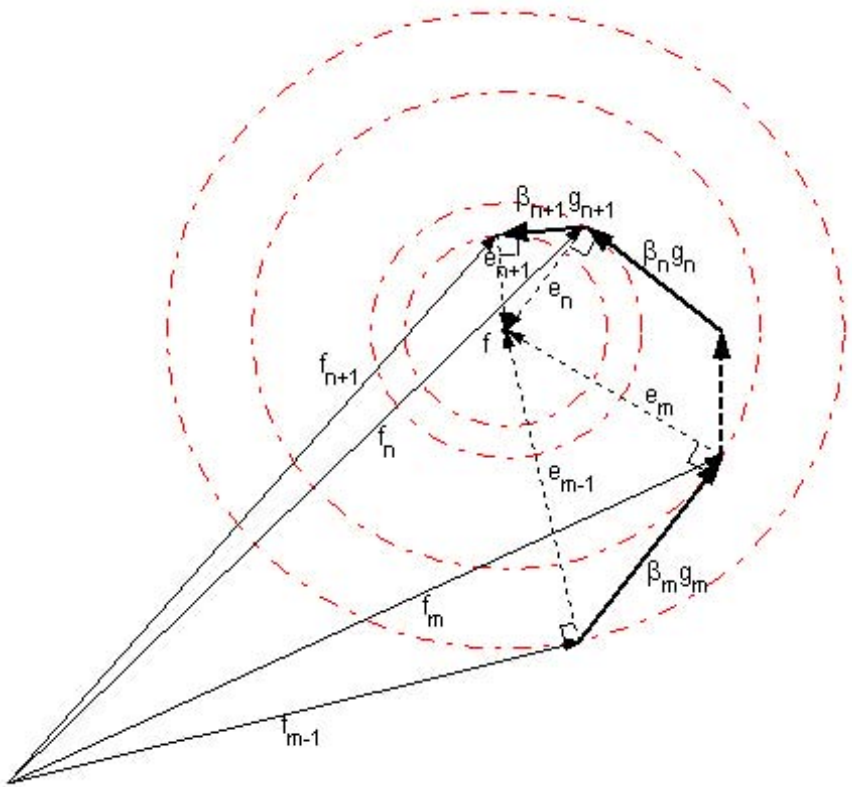


Figure 3: Feedforward Network Architecture: any type of nonlinear piecewise continuous $G(\mathbf{a}_i, b_i, \mathbf{x})$.

New Learning Theory

1. Given a SLFN with a type of nonconstant piecewise continuous hidden nodes $G(\mathbf{x}, \mathbf{a}, b)$, if any continuous target function $f(\mathbf{x})$ can be approximated by such SLFNs with appropriate hidden node parameters, then there is no need to find an algorithm to tune the hidden node parameters.
2. Instead, given any positive value ϵ, for any continuous target function f and any randomly generated sequence $\{(\mathbf{a}_i, b_i)_{i=1}^L\}$, there exists an integer $L_0 > 0$ such that when $L > L_0$, $\left\| f(\mathbf{x}) - \sum_{n=1}^L \beta_n g_n \right\| < \epsilon$ holds with probability one if $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$, $g_n = G(\mathbf{a}_n, b_n, \mathbf{x})$, $i = 1, \cdots, L$.
3. Thus, for basic ELM with the fixed network architecture and L random hidden nodes, $\lim_{L \to +\infty} \left\| f(\mathbf{x}) - \sum_{i=1}^L \boldsymbol{\beta}_i g_i \right\| = 0$ where the output weights $\boldsymbol{\beta}_i$'s are determined by ordinary least square.

Essence of ELM

Hidden node parameters $(\mathbf{a}_i, b_i)_{i=1}^L$ are not only independent of target functions $f(\mathbf{x})$ but also of training samples.

ELM Learning Theory

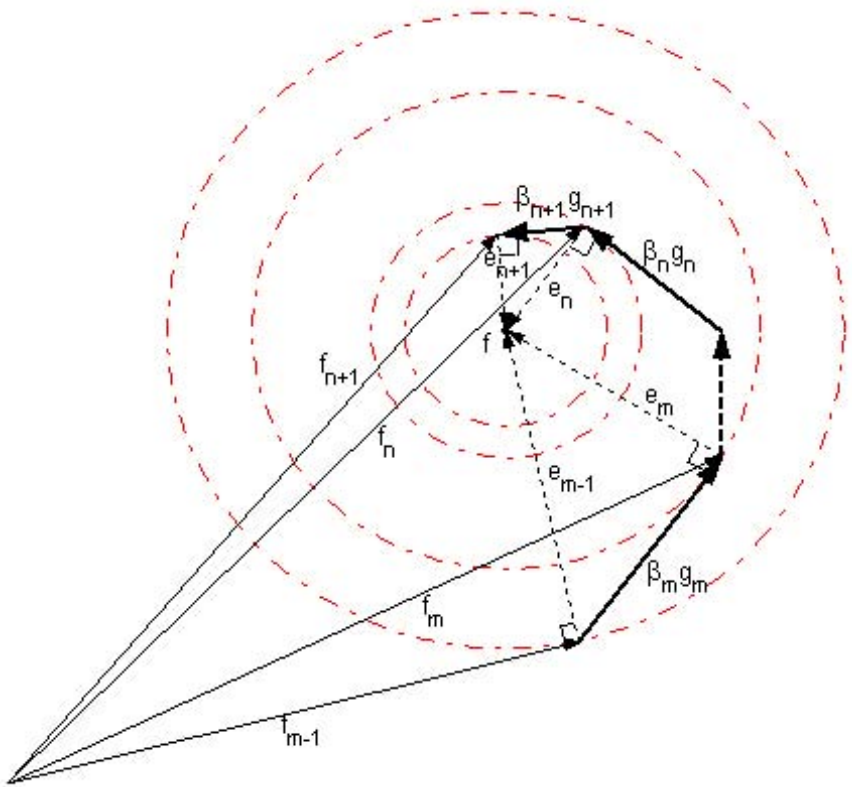


Figure 3: Feedforward Network Architecture: any type of nonlinear piecewise continuous $G(\mathbf{a}_i, b_i, \mathbf{x})$.

New Learning Theory

1. Given a SLFN with a type of nonconstant piecewise continuous hidden nodes $G(\mathbf{x}, \mathbf{a}, b)$, if any continuous target function $f(\mathbf{x})$ can be approximated by such SLFNs with appropriate hidden node parameters, then there is no need to find an algorithm to tune the hidden node parameters.
2. Instead, given any positive value ϵ, for any continuous target function f and any randomly generated sequence $\{(\mathbf{a}_i, b_i)_{i=1}^L\}$, there exists an integer $L_0 > 0$ such that when $L > L_0$, $\left\|f(\mathbf{x}) - \sum_{n=1}^L \beta_n g_n\right\| < \epsilon$ holds with probability one if $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$, $g_n = G(\mathbf{a}_n, b_n, \mathbf{x})$, $i = 1, \cdots, L$.
3. Thus, for basic ELM with the fixed network architecture and L random hidden nodes, $\lim_{L \to +\infty} \left\|f(\mathbf{x}) - \sum_{i=1}^L \boldsymbol{\beta}_i g_i\right\| = 0$ where the output weights $\boldsymbol{\beta}_i$'s are determined by ordinary least square.

Essence of ELM

Hidden node parameters $(\mathbf{a}_i, b_i)_{i=1}^L$ are not only independent of target functions $f(\mathbf{x})$ but also of training samples.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and Baum's Work

1. Baum (1988): (seen from simulations) one may fix the weights of the connections on one level and simply adjust the connections on the other level and no (significant) gain is possible by using an algorithm able to adjust the weights on both levels simultaneously.
2. However, Baum did not discuss whether all the hidden node biases b_i should be set with the same value. Baum did not discuss either whether the hidden node biases b_i should be tuned or not. ELM theory states that the hidden node parameters are independent of the training data, which was not found in Baum (1988).
3. Baum (1988) did not study RBF network and kernel learning, while ELM work for all these cases.
4. Baum (1988) did not give any theoretical analysis, let alone the proof of universal approximation capability of ELM.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and Baum's Work

1. Baum (1988): (seen from simulations) one may fix the weights of the connections on one level and simply adjust the connections on the other level and no (significant) gain is possible by using an algorithm able to adjust the weights on both levels simultaneously.
2. However, Baum did not discuss whether all the hidden node biases b_i should be set with the same value. Baum did not discuss either whether the hidden node biases b_i should be tuned or not. ELM theory states that the hidden node parameters are independent of the training data, which was not found in Baum (1988).
3. Baum (1988) did not study RBF network and kernel learning, while ELM work for all these cases.
4. Baum (1988) did not give any theoretical analysis, let alone the proof of universal approximation capability of ELM.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and Baum's Work

1. Baum (1988): (seen from simulations) one may fix the weights of the connections on one level and simply adjust the connections on the other level and no (significant) gain is possible by using an algorithm able to adjust the weights on both levels simultaneously.
2. However, Baum did not discuss whether all the hidden node biases b_i should be set with the same value. Baum did not discuss either whether the hidden node biases b_i should be tuned or not. ELM theory states that the hidden node parameters are independent of the training data, which was not found in Baum (1988).
3. Baum (1988) did not study RBF network and kernel learning, while ELM work for all these cases.
4. Baum (1988) did not give any theoretical analysis, let alone the proof of universal approximation capability of ELM.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and Baum's Work

1. Baum (1988): (seen from simulations) one may fix the weights of the connections on one level and simply adjust the connections on the other level and no (significant) gain is possible by using an algorithm able to adjust the weights on both levels simultaneously.
2. However, Baum did not discuss whether all the hidden node biases b_i should be set with the same value. Baum did not discuss either whether the hidden node biases b_i should be tuned or not. ELM theory states that the hidden node parameters are independent of the training data, which was not found in Baum (1988).
3. Baum (1988) did not study RBF network and kernel learning, while ELM work for all these cases.
4. Baum (1988) did not give any theoretical analysis, let alone the proof of universal approximation capability of ELM.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and Baum's Work

1. Baum (1988): (seen from simulations) one may fix the weights of the connections on one level and simply adjust the connections on the other level and no (significant) gain is possible by using an algorithm able to adjust the weights on both levels simultaneously.
2. However, Baum did not discuss whether all the hidden node biases b_i should be set with the same value. Baum did not discuss either whether the hidden node biases b_i should be tuned or not. ELM theory states that the hidden node parameters are independent of the training data, which was not found in Baum (1988).
3. Baum (1988) did not study RBF network and kernel learning, while ELM work for all these cases.
4. Baum (1988) did not give any theoretical analysis, let alone the proof of universal approximation capability of ELM.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RBF Networks

1. The conventional RBF network (Lowe 1988,Lowe 1989): focus on a specific RBF network with the same impact factor b assigned to all the RBF hidden nodes: $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$, where the centers $\mathbf{a}_i$ can be randomly selected from the training data instead of tuning, but the impact factor b_i of RBF hidden nodes is not randomly selected and usually determined by users. One of RBF networks interested by ELM is $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ where the RBF hidden nodes are not requested to have the same impact factors b_i.
2. RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$ (studied by Lowe 1988,Lowe 1989) with randomly generated centers $\mathbf{a}_i$ and randomly generated same values of impact factors b in fact does not generally have the universal approximation capability, in contrast, RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ (in ELM) with randomly generated centers $\mathbf{a}_i$ and randomly generated impact factors b_i (with different values) does generally have the universal approximation capability.
3. ELM works for different type of hidden nodes including different type of RBF nodes (not limited to $g(b\|\mathbf{x} - \mathbf{a}_i\|)$), additive nodes, kernels, etc while conventional RBF networks (Lowe1988,Lowe1989) only work for specific type of RBF networks with single impact factor value for all RBF nodes.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RBF Networks

1. The conventional RBF network (Lowe 1988,Lowe 1989): focus on a specific RBF network with the same impact factor b assigned to all the RBF hidden nodes: $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$, where the centers $\mathbf{a}_i$ can be randomly selected from the training data instead of tuning, but the impact factor b_i of RBF hidden nodes is not randomly selected and usually determined by users. One of RBF networks interested by ELM is $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ where the RBF hidden nodes are not requested to have the same impact factors b_i.
2. RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$ (studied by Lowe 1988,Lowe 1989) with randomly generated centers $\mathbf{a}_i$ and randomly generated same values of impact factors b in fact does not generally have the universal approximation capability, in contrast, RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ (in ELM) with randomly generated centers $\mathbf{a}_i$ and randomly generated impact factors b_i (with different values) does generally have the universal approximation capability.
3. ELM works for different type of hidden nodes including different type of RBF nodes (not limited to $g(b\|\mathbf{x} - \mathbf{a}_i\|)$), additive nodes, kernels, etc while conventional RBF networks (Lowe1988,Lowe1989) only work for specific type of RBF networks with single impact factor value for all RBF nodes.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RBF Networks

1. The conventional RBF network (Lowe 1988,Lowe 1989): focus on a specific RBF network with the same impact factor b assigned to all the RBF hidden nodes: $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$, where the centers $\mathbf{a}_i$ can be randomly selected from the training data instead of tuning, but the impact factor b_i of RBF hidden nodes is not randomly selected and usually determined by users. One of RBF networks interested by ELM is $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ where the RBF hidden nodes are not requested to have the same impact factors b_i.
2. RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$ (studied by Lowe 1988,Lowe 1989) with randomly generated centers $\mathbf{a}_i$ and randomly generated same values of impact factors b in fact does not generally have the universal approximation capability, in contrast, RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ (in ELM) with randomly generated centers $\mathbf{a}_i$ and randomly generated impact factors b_i (with different values) does generally have the universal approximation capability.
3. ELM works for different type of hidden nodes including different type of RBF nodes (not limited to $g(b\|\mathbf{x} - \mathbf{a}_i\|)$), additive nodes, kernels, etc while conventional RBF networks (Lowe1988,Lowe1989) only work for specific type of RBF networks with single impact factor value for all RBF nodes.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RBF Networks

1. The conventional RBF network (Lowe 1988,Lowe 1989): focus on a specific RBF network with the same impact factor b assigned to all the RBF hidden nodes: $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$, where the centers $\mathbf{a}_i$ can be randomly selected from the training data instead of tuning, but the impact factor b_i of RBF hidden nodes is not randomly selected and usually determined by users. One of RBF networks interested by ELM is $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ where the RBF hidden nodes are not requested to have the same impact factors b_i.
2. RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b\|\mathbf{x} - \mathbf{a}_i\|)$ (studied by Lowe 1988,Lowe 1989) with randomly generated centers $\mathbf{a}_i$ and randomly generated same values of impact factors b in fact does not generally have the universal approximation capability, in contrast, RBF networks $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g(b_i\|\mathbf{x} - \mathbf{a}_i\|)$ (in ELM) with randomly generated centers $\mathbf{a}_i$ and randomly generated impact factors b_i (with different values) does generally have the universal approximation capability.
3. ELM works for different type of hidden nodes including different type of RBF nodes (not limited to $g(b\|\mathbf{x} - \mathbf{a}_i\|)$), additive nodes, kernels, etc while conventional RBF networks (Lowe1988,Lowe1989) only work for specific type of RBF networks with single impact factor value for all RBF nodes.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RVFL

1. In a random vector version of the functional-link (RVFL) model (Igelnik 1995), the input weights $\mathbf{a}_i$ are "uniformly" drawn from a probabilistic space $V_\alpha^d = [0, \alpha\Omega] \times [-\alpha\Omega, \alpha\Omega]^{d-1}$ (d: the input dimension). The hidden node biases b_i depend on the weights $\mathbf{a}_i$ and some other parameters $\mathbf{y}_i$ and u_i: $b_i = -(\alpha\mathbf{a}_i \cdot \mathbf{y}_i + u_i)$, where $\mathbf{y}_i$ and u_i are randomly generated from $[0, 1]^d$ and $[-2\Omega, 2\Omega]$. α and Ω have to be determined in the learning stage and depends on the training data distribution.
2. In ELM, the hidden node parameters $(\mathbf{a}_i, b_i)$ are not only independent of the training data but also of each other.
3. In ELM, $\mathbf{a}_i$'s and b_i's are independent of each other.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RVFL

1. In a random vector version of the functional-link (RVFL) model (Igelnik 1995), the input weights $\mathbf{a}_i$ are "uniformly" drawn from a probabilistic space $V_\alpha^d = [0, \alpha\Omega] \times [-\alpha\Omega, \alpha\Omega]^{d-1}$ (d: the input dimension). The hidden node biases b_i depend on the weights $\mathbf{a}_i$ and some other parameters $\mathbf{y}_i$ and u_i: $b_i = -(\alpha \mathbf{a}_i \cdot \mathbf{y}_i + u_i)$, where $\mathbf{y}_i$ and u_i are randomly generated from $[0, 1]^d$ and $[-2\Omega, 2\Omega]$. α and Ω have to be determined in the learning stage and depends on the training data distribution.
2. In ELM, the hidden node parameters $(\mathbf{a}_i, b_i)$ are not only independent of the training data but also of each other.
3. In ELM, $\mathbf{a}_i$'s and b_i's are independent of each other.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RVFL

1. In a random vector version of the functional-link (RVFL) model (Igelnik 1995), the input weights $\mathbf{a}_i$ are "uniformly" drawn from a probabilistic space $V_\alpha^d = [0, \alpha\Omega] \times [-\alpha\Omega, \alpha\Omega]^{d-1}$ (d: the input dimension). The hidden node biases b_i depend on the weights $\mathbf{a}_i$ and some other parameters $\mathbf{y}_i$ and u_i: $b_i = -(\alpha \mathbf{a}_i \cdot \mathbf{y}_i + u_i)$, where $\mathbf{y}_i$ and u_i are randomly generated from $[0, 1]^d$ and $[-2\Omega, 2\Omega]$. α and Ω have to be determined in the learning stage and depends on the training data distribution.
2. In ELM, the hidden node parameters $(\mathbf{a}_i, b_i)$ are not only independent of the training data but also of each other.
3. In ELM, $\mathbf{a}_i$'s and b_i's are independent of each other.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Differences Between ELM and Semi-Random Methods

Difference Between ELM and RVFL

1. In a random vector version of the functional-link (RVFL) model (Igelnik 1995), the input weights $\mathbf{a}_i$ are "uniformly" drawn from a probabilistic space $V_\alpha^d = [0, \alpha\Omega] \times [-\alpha\Omega, \alpha\Omega]^{d-1}$ (d: the input dimension). The hidden node biases b_i depend on the weights $\mathbf{a}_i$ and some other parameters $\mathbf{y}_i$ and u_i: $b_i = -(\alpha\mathbf{a}_i \cdot \mathbf{y}_i + u_i)$, where $\mathbf{y}_i$ and u_i are randomly generated from $[0, 1]^d$ and $[-2\Omega, 2\Omega]$. α and Ω have to be determined in the learning stage and depends on the training data distribution.
2. In ELM, the hidden node parameters $(\mathbf{a}_i, b_i)$ are not only independent of the training data but also of each other.
3. In ELM, $\mathbf{a}_i$'s and b_i's are independent of each other.

G.-B. Huang, et al., "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576-583, 2008.

Incremental Extreme Learning Machine (I-ELM)

I-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum node number $L_{\max}$ and expected learning accuracy ϵ,

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

   while $L < L_{\max}$ and $||E|| > \epsilon$

   endwhile

where $H_L = [h(1), \cdots, h(N)]^T$ is the activation vector of the new node L for all the N training samples and $E = [e(1), \cdots, e(N)]^T$ is the residual vector. $E \cdot H_L^T \approx \langle e_{L-1}, g_L \rangle$ and $H_L \cdot H_L^T \approx \|g_L\|^2$.

Incremental Extreme Learning Machine (I-ELM)

I-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum node number $L_{\max}$ and expected learning accuracy ϵ,

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

   while $L < L_{\max}$ and $||E|| > \epsilon$

   - Increase by 1 the number of hidden nodes L: $L = L + 1$.
   - Assign random hidden node parameters $\mathbf{a}_L$ and b_L for the new hidden node L.
   - Calculate the output weight β_L for the new hidden node L.
   - Calculate the residual error after adding the new hidden node L.

   endwhile

where $H_L = [h(1), \cdots, h(N)]^T$ is the activation vector of the new node L for all the N training samples and $E = [e(1), \cdots, e(N)]^T$ is the residual vector. $E \cdot H_L^T \approx \langle e_{L-1}, g_L \rangle$ and $H_L \cdot H_L^T \approx \|g_L\|^2$.

Incremental Extreme Learning Machine (I-ELM)

I-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum node number $L_{\max}$ and expected learning accuracy ϵ,

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

   while $L < L_{\max}$ and $\|E\| > \epsilon$

   - Increase by 1 the number of hidden nodes L: $L = L + 1$.
   - Assign random hidden node parameter $(\mathbf{a}_L, b_L)$ for new hidden node L.
   - Calculate the output weight β_L for the new hidden node: $\beta_L = \frac{E \cdot H_L^T}{H_L \cdot H_L^T} \approx \frac{\langle e_{L-1}, g_L \rangle}{\|g_L\|^2}$
   - Calculate the residual error after adding the new hidden node L: $E = E - \beta_L \cdot H_L$

   endwhile

where $H_L = [h(1), \cdots, h(N)]^T$ is the activation vector of the new node L for all the N training samples and $E = [e(1), \cdots, e(N)]^T$ is the residual vector. $E \cdot H_L^T \approx \langle e_{L-1}, g_L \rangle$ and $H_L \cdot H_L^T \approx \|g_L\|^2$.

Incremental Extreme Learning Machine (I-ELM)

I-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum node number $L_{\max}$ and expected learning accuracy ϵ,

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

   while $L < L_{\max}$ and $\|E\| > \epsilon$

   - Increase by 1 the number of hidden nodes L: $L = L + 1$.
   - Assign random hidden node parameter $(\mathbf{a}_L, b_L)$ for new hidden node L.
   - Calculate the output weight β_L for the new hidden node: $\beta_L = \frac{E \cdot H_L^T}{H_L \cdot H_L^T} \approx \frac{\langle e_{L-1}, g_L \rangle}{\|g_L\|^2}$
   - Calculate the residual error after adding the new hidden node L: $E = E - \beta_L \cdot H_L$

   endwhile

where $H_L = [h(1), \cdots, h(N)]^T$ is the activation vector of the new node L for all the N training samples and $E = [e(1), \cdots, e(N)]^T$ is the residual vector. $E \cdot H_L^T \approx \langle e_{L-1}, g_L \rangle$ and $H_L \cdot H_L^T \approx \|g_L\|^2$.

Incremental Extreme Learning Machine (I-ELM)

I-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum node number $L_{\max}$ and expected learning accuracy ϵ,

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

   while $L < L_{\max}$ and $\|E\| > \epsilon$

   - Increase by 1 the number of hidden nodes L: $L = L + 1$.
   - Assign random hidden node parameter $(\mathbf{a}_L, b_L)$ for new hidden node L.
   - Calculate the output weight β_L for the new hidden node: $\beta_L = \frac{E \cdot H_L^T}{H_L \cdot H_L^T} \approx \frac{\langle e_{L-1}, g_L \rangle}{\|g_L\|^2}$
   - Calculate the residual error after adding the new hidden node L: $E = E - \beta_L \cdot H_L$

   endwhile

where $H_L = [h(1), \cdots, h(N)]^T$ is the activation vector of the new node L for all the N training samples and $E = [e(1), \cdots, e(N)]^T$ is the residual vector. $E \cdot H_L^T \approx \langle e_{L-1}, g_L \rangle$ and $H_L \cdot H_L^T \approx \|g_L\|^2$.

Incremental Extreme Learning Machine (I-ELM)

I-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum node number $L_{\max}$ and expected learning accuracy ϵ,

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

   while $L < L_{\max}$ and $\|E\| > \epsilon$

   - Increase by 1 the number of hidden nodes L: $L = L + 1$.
   - Assign random hidden node parameter $(\mathbf{a}_L, b_L)$ for new hidden node L.
   - Calculate the output weight β_L for the new hidden node: $\beta_L = \frac{E \cdot H_L^T}{H_L \cdot H_L^T} \approx \frac{\langle e_{L-1}, g_L \rangle}{\|g_L\|^2}$
   - Calculate the residual error after adding the new hidden node L: $E = E - \beta_L \cdot H_L$

   endwhile

where $H_L = [h(1), \cdots, h(N)]^T$ is the activation vector of the new node L for all the N training samples and $E = [e(1), \cdots, e(N)]^T$ is the residual vector. $E \cdot H_L^T \approx \langle e_{L-1}, g_L \rangle$ and $H_L \cdot H_L^T \approx \|g_L\|^2$.

Incremental Extreme Learning Machine (I-ELM)

I-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum node number $L_{\max}$ and expected learning accuracy ϵ,

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

   while $L < L_{\max}$ and $\|E\| > \epsilon$

   - Increase by 1 the number of hidden nodes L: $L = L + 1$.
   - Assign random hidden node parameter $(\mathbf{a}_L, b_L)$ for new hidden node L.
   - Calculate the output weight β_L for the new hidden node: $\beta_L = \frac{E \cdot H_L^T}{H_L \cdot H_L^T} \approx \frac{\langle e_{L-1}, g_L \rangle}{\|g_L\|^2}$
   - Calculate the residual error after adding the new hidden node L: $E = E - \beta_L \cdot H_L$

   endwhile

where $H_L = [h(1), \cdots, h(N)]^T$ is the activation vector of the new node L for all the N training samples and $E = [e(1), \cdots, e(N)]^T$ is the residual vector. $E \cdot H_L^T \approx \langle e_{L-1}, g_L \rangle$ and $H_L \cdot H_L^T \approx \|g_L\|^2$.

Performance of I-ELM with RBF hidden nodes

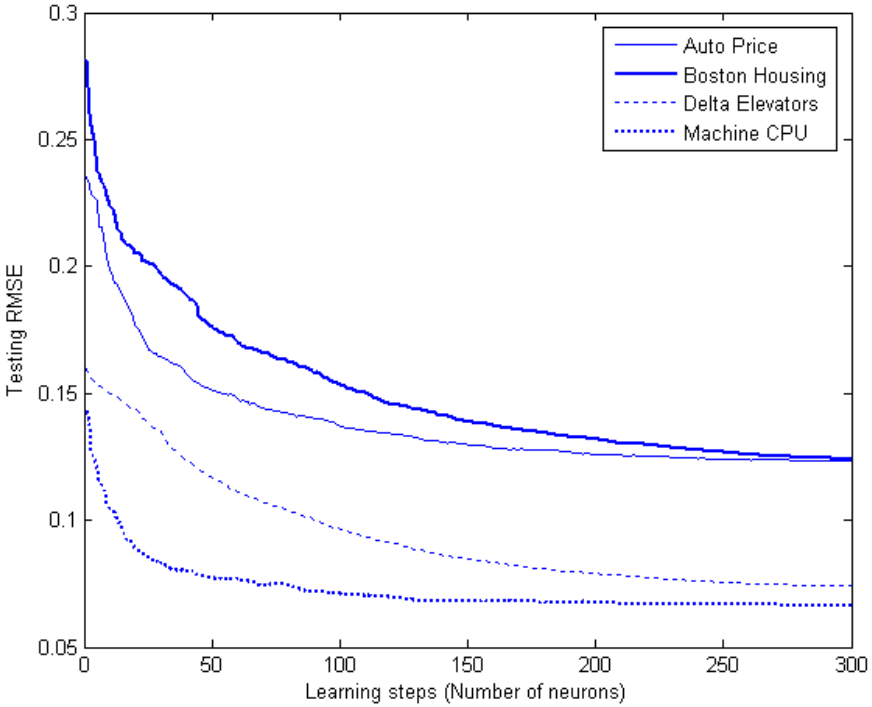


Figure 4: Average testing RMSE

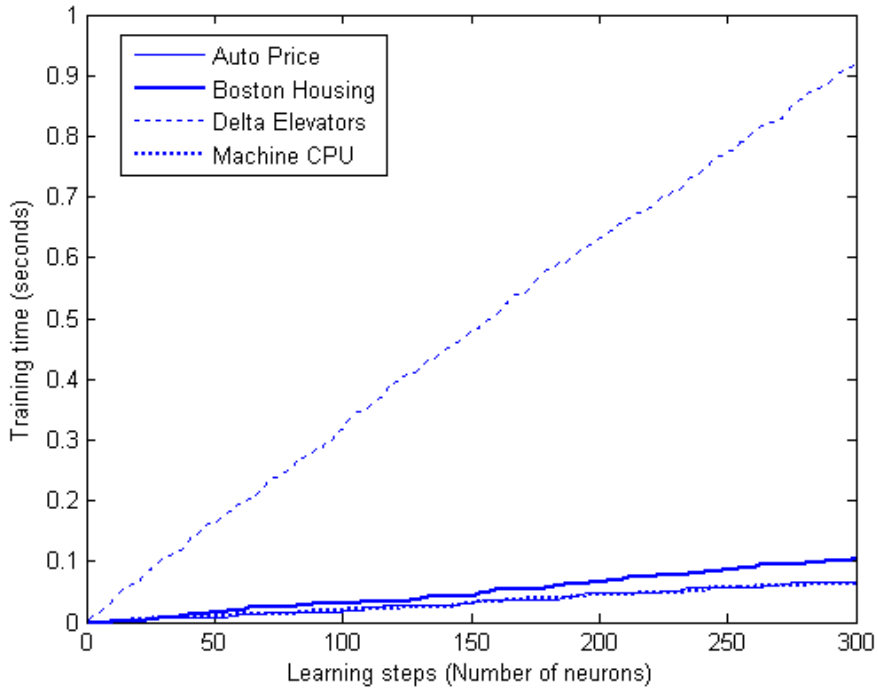


Figure 5: Average training time (seconds)

G.-B. Huang, et al., "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879-892, 2006.

Real-World Regression Problems

Problems	I-ELM			RAN	MRAN
	Sigmoid	RBF	Sin		
Abalone	0.0920	0.0938	**0.0886**	0.1183	0.0906
Auto Price	**0.0977**	0.1261	0.1162	0.1418	0.1373
Boston Housing	**0.1167**	0.1320	0.1404	0.1474	0.1321
California Housing	0.1683	0.1731	0.1550	0.1506	**0.1480**
Census (House8L)	0.0923	0.0922	**0.0842**	0.1061	0.0903
Delta Ailerons	**0.0525**	0.0632	0.0635	0.1018	0.0618
Delta Elevators	0.0740	0.0790	**0.0739**	0.1322	0.0807
Machine CPU	**0.0504**	0.0674	0.0665	0.1069	0.1068

Table 1: Average testing RMSE of different algorithms. (I-ELM with 200 hidden nodes)

Real-World Regression Problems

Problems	I-ELM			RAN	MRAN
	Sigmoid	RBF	Sin		
Abalone	0.0046	0.0053	0.0049	0.0076	0.0065
Auto Price	0.0069	0.0255	0.0179	0.0261	0.0381
Boston Housing	0.0112	0.0126	0.0114	0.0177	0.0140
California Housing	0.0049	0.0081	0.0052	0.0035	0.0030
Census (House8L)	0.0023	0.0029	0.0015	0.0038	0.0042
Delta Ailerons	0.0078	0.0116	0.0090	0.0083	0.0050
Delta Elevators	0.0126	0.0123	0.0065	0.0130	0.0068
Machine CPU	0.0079	0.0177	0.0278	0.0246	0.0367

Table 2: Standard deviations (Dev) of testing RMSE of different algorithms. (I-ELM with 200 hidden nodes)

Real-World Regression Problems

Problems	Training time of I-ELM			Training Time		# nodes	
	Sigmoid	RBF	Sin	RAN	MRAN	RAN	MRAN
Abalone	0.2214	0.5030	0.1778	39.928	255.84	186.3	67.7
Auto Price	0.0329	0.0468	0.0188	0.3565	2.5015	23.8	22.5
Boston Housing	0.0515	0.0657	0.0470	2.0940	22.767	40.5	36.2
California Housing	0.5448	1.3656	0.3872	3301.7	2701.1	4883.0	93.0
Census (House8L)	0.8667	1.7928	0.5194	5399.0	3805.3	6393.2	77.3
Delta Ailerons	0.2620	0.4327	0.1715	237.96	175.07	1118.1	76.6
Delta Elevators	0.2708	0.6321	0.2261	661.78	331.75	2417.4	76.8
Machine CPU	0.0234	0.0447	0.0297	0.1735	0.2454	6.9	7.0

Table 3: Training time (seconds) and network complexity comparison of different algorithms

Real-World Regression Problems

Problems	I-ELM		SGBP ($\lambda = 1$)		SVR	
	Mean	Dev	Mean	Dev	Mean	Dev
Abalone	0.0878	0.0032	0.1175	0.0095	0.0846	0.0013
Auto Price	**0.0883**	0.0036	0.2383	0.0587	0.1052	0.0040
Boston Housing	**0.1095**	0.0090	0.1882	0.0243	0.1155	0.0079
California Housing	0.1555	0.0021	0.1579	0.0033	**0.1311**	0.0011
Census (House8L)	0.0871	0.0021	0.0866	0.0025	**0.0683**	0.0013
Delta Ailerons	0.0472	0.0049	0.0459	0.0033	0.0467	0.0010
Delta Elevators	0.0639	0.0067	0.0653	0.0019	0.0603	0.0005
Machine CPU	**0.0491**	0.0089	0.1988	0.0429	0.0620	0.0180

Table 4: Performance comparison (testing RMSE and the corresponding standard deviation) of I-ELM (with 500 random sigmoid hidden nodes), stochastic gradient descent BP (SGBP), and SVR.

Real-World Regression Problems

Problems	I-ELM[a]	SGBP[a] ($\lambda = 1$)		SVR[b]		
	Time (s)	Time (s)	# Nodes	Time (s)	# SVs	(C, γ)
Abalone	0.5560	0.4406	10	1.6123	309.84	$(2^4, 2^{-6})$
Auto Price	0.0954	0.0154	15	0.0042	21.25	$(2^8, 2^{-5})$
Boston Housing	0.1419	0.0579	10	0.0494	46.44	$(2^4, 2^{-3})$
California Housing	1.3763	2.0307	10	74.184	2189.2	$(2^3, 2^1)$
Census (House8L)	1.7295	2.7814	30	11.251	810.24	$(2^1, 2^{-1})$
Delta Ailerons	0.7058	0.6610	10	0.6726	82.44	$(2^3, 2^{-3})$
Delta Elevators	0.7296	0.8830	10	1.1210	260.38	$(2^0, 2^{-2})$
Machine CPU	0.0765	0.0206	10	0.0018	7.8	$(2^6, 2^{-4})$

[a] run in MATLAB environment. [b] run in C executable environment.

Table 5: Performance comparison (training time (seconds)) of I-ELM (with 500 random sigmoid hidden nodes), stochastic gradient descent BP (SGBP), and SVR.

Real-World Regression Problems

Problems	Testing RMSE		Dev of Testing RMSE		Training Time (s)	
	I-ELM (Threshold)	SGBP ($\lambda = 10$)	I-ELM (Threshold)	SGBP ($\lambda = 10$)	I-ELM (Threshold)	SGBP ($\lambda = 10$)
Abalone	**0.0951**	0.1332	0.0142	0.0102	0.2908	0.4313
Auto Price	**0.1141**	0.3209	0.0130	0.0665	0.0735	0.0172
Boston Housing	**0.1346**	0.2196	0.0104	0.0279	0.0907	0.0548
California Housing	0.1828	0.1806	0.0179	0.0226	0.8186	1.9548
Census (House8L)	**0.0941**	0.1032	0.0062	0.0068	1.0117	2.7359
Delta Ailerons	0.0790	**0.0400**	0.0397	0.0055	0.3550	0.6375
Delta Elevators	**0.0713**	0.0895	0.0110	0.0090	0.5102	0.8970
Machine CPU	**0.0739**	0.2281	0.0140	0.0479	0.0658	0.0215

Table 6: Performance comparison between the approximated threshold network ($\lambda = 10$) trained by stochastic gradient descent BP (SGBP) and the true threshold networks trained by I-ELM with 500 threshold nodes: $g(x) = -1_{x<0} + 1_{x\geq 0}$.

Enhanced Incremental ELM (EI-ELM)

New Convergence Theorem

Given a SLFN with any nonconstant piecewise continuous hidden nodes $G(\mathbf{x}, \mathbf{a}, b)$, if $span\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{C}^d \times C\}$ is dense in L^2, for any continuous target function f and any randomly generated function sequence $\{g_n\}$ and any positive integer k, $\lim_{n\to\infty} \|f - f_n^*\| = 0$ holds with probability one if

$$\beta_n^* = \frac{\left\langle e_{n-1}^*, g_n^* \right\rangle}{\|g_n^*\|^2} \tag{3}$$

where $f_n^* = \sum_{i=1}^{n} \beta_i^* g_i^*$, $e_n^* = f - f_n^*$ and $g_n^* = \{g_i | \min_{(n-1)k+1 \le i \le nk} \|(f - f_{n-1}^*) - \beta_n g_i\|\}$.

G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine,"

Neurocomputing, vol. 71, pp. 3460-3468, 2008.

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

for

endfor

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

+ Increase by 1 the number of hidden nodes L: $L = L + 1$.
+ **for** $i = 1 : k$
  - Assign random parameters $(\mathbf{a}_i, b_i)$ for the i-th candidate node.
  - Calculate the output weight β_i for the i-th candidate node.
  - Calculate the residual error after adding the i-th candidate node: $E_{(i)} = E - \beta_i \mathbf{h}_i$.

endfor

+ Let $i^* = \{i | \min_{1 \le i \le k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\beta_L = \beta_{(i^*)}$.

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

\+ Increase by 1 the number of hidden nodes L: $L = L + 1$.

\+ for $i = 1 : k$

- Assign random parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new hidden node i according to any continuous sampling distribution probability.
- Calculate the output weight $\beta_{(i)}$ for the new hidden node i.
- Calculate the residual error after adding the new hidden node i.

endfor

\+ Let $i^* = \{i | \min_{1 \le i \le k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\beta_L = \beta_{(i^*)}$.

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

+ Increase by 1 the number of hidden nodes L: $L = L + 1$.
+ **for** $i = 1 : k$
  - Assign random parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new hidden node L according to any continuous sampling distribution probability.
  - Calculate the output weight $\beta_{(i)}$ for the new hidden node: $\beta_{(i)} = \frac{E \cdot H_{(i)}^T}{H_{(i)} \cdot H_{(i)}^T}$
  - Calculate the residual error after adding the new hidden node L: $E_{(i)} = E - \beta_{(i)} \cdot H_{(i)}$

  endfor
+ Let $i^* = \{i | \min_{1 \le i \le k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\beta_L = \beta_{(i^*)}$.

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

+ Increase by 1 the number of hidden nodes L: $L = L + 1$.
+ **for** $i = 1 : k$
  - Assign random parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new hidden node L according to any continuous sampling distribution probability.
  - Calculate the output weight $\beta_{(i)}$ for the new hidden node: $\beta_{(i)} = \frac{E \cdot H_{(i)}^T}{H_{(i)} \cdot H_{(i)}^T}$
  - Calculate the residual error after adding the new hidden node L: $E_{(i)} = E - \beta_{(i)} \cdot H_{(i)}$

  endfor
+ Let $i^* = \{i | \min_{1 \le i \le k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\beta_L = \beta_{(i^*)}$.

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

+ Increase by 1 the number of hidden nodes L: $L = L + 1$.
+ **for** $i = 1 : k$
  - Assign random parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new hidden node L according to any continuous sampling distribution probability.
  - Calculate the output weight $\beta_{(i)}$ for the new hidden node: $\beta_{(i)} = \frac{E \cdot H_{(i)}^T}{H_{(i)} \cdot H_{(i)}^T}$
  - Calculate the residual error after adding the new hidden node L: $E_{(i)} = E - \beta_{(i)} \cdot H_{(i)}$

  endfor
+ Let $i^* = \{i | \min_{1 \le i \le k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\beta_L = \beta_{(i^*)}$.

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

+ Increase by 1 the number of hidden nodes L: $L = L + 1$.
+ **for** $i = 1 : k$
  - Assign random parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new hidden node L according to any continuous sampling distribution probability.
  - Calculate the output weight $\beta_{(i)}$ for the new hidden node: $\beta_{(i)} = \frac{E \cdot H_{(i)}^T}{H_{(i)} \cdot H_{(i)}^T}$
  - Calculate the residual error after adding the new hidden node L: $E_{(i)} = E - \beta_{(i)} \cdot H_{(i)}$

  endfor
+ Let $i^* = \{i | \min_{1 \le i \le k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\beta_L = \beta_{(i^*)}$.

endwhile

Enhanced Incremental ELM (EI-ELM)

EI-ELM Algorithm

1. **Initialization:** Let $L = 0$ and residual error $E = t$, where $t = [t_1, \cdots, t_N]^T$.
2. **Learning step:**

while $L < L_{\max}$ and $\|E\| > \epsilon$

+ Increase by 1 the number of hidden nodes L: $L = L + 1$.
+ **for** $i = 1 : k$
  - Assign random parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new hidden node L according to any continuous sampling distribution probability.
  - Calculate the output weight $\beta_{(i)}$ for the new hidden node: $\beta_{(i)} = \frac{E \cdot H_{(i)}^T}{H_{(i)} \cdot H_{(i)}^T}$
  - Calculate the residual error after adding the new hidden node L: $E_{(i)} = E - \beta_{(i)} \cdot H_{(i)}$

  endfor
+ Let $i^* = \{i | \min_{1 \le i \le k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\beta_L = \beta_{(i^*)}$.

endwhile

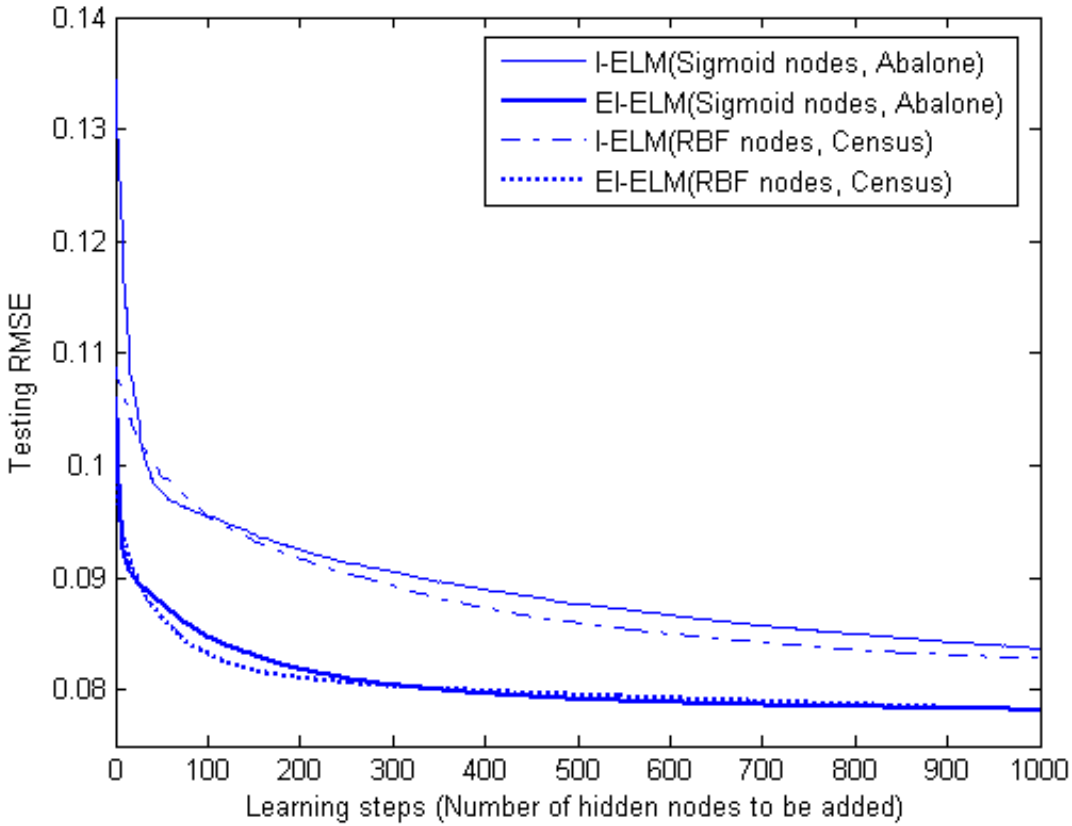


Figure 6: The testing error updating curves of EI-ELM and I-ELM

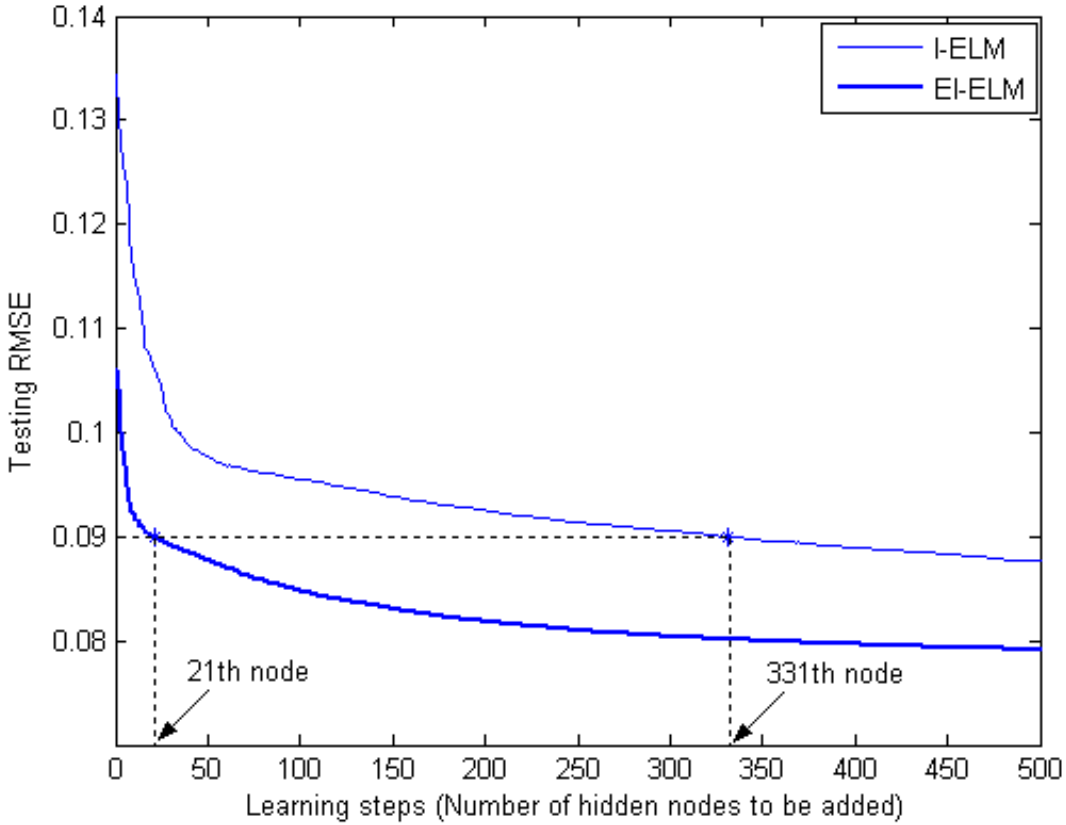


Figure 7: Testing RMSE performance comparison between EI-ELM and I-ELM (with Sigmoid hidden nodes) for Abalone case

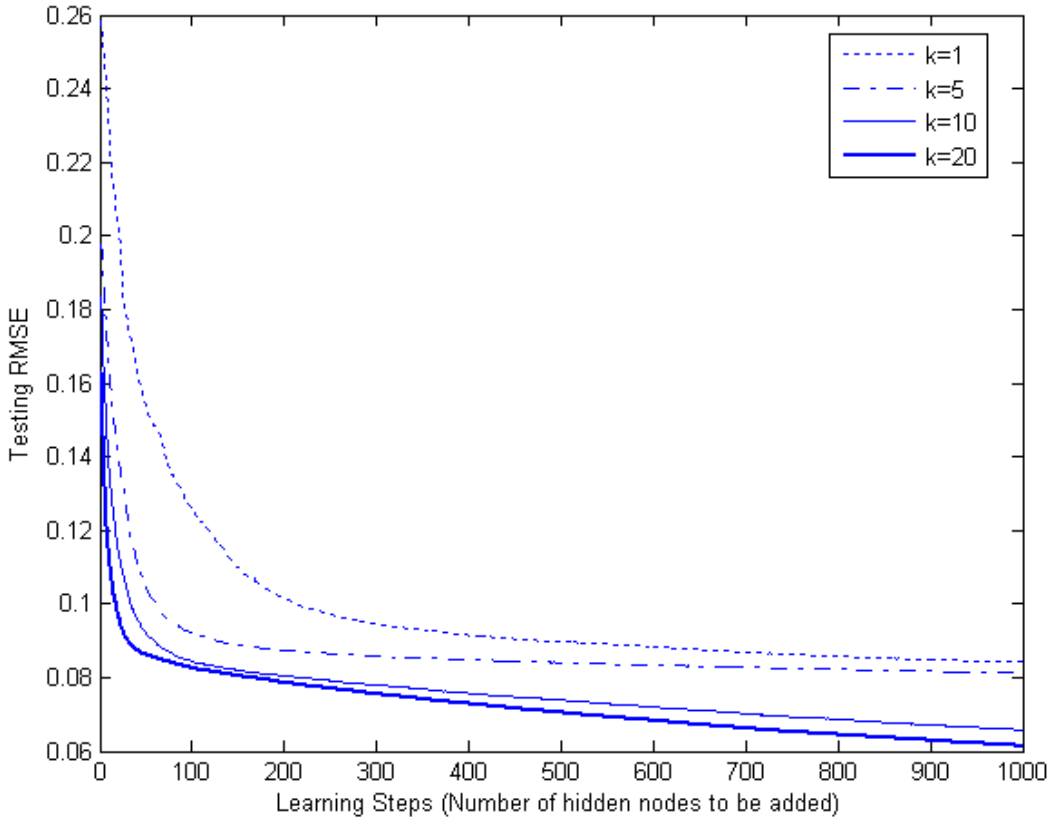


Figure 8: Testing RMSE updating progress with new hidden nodes added and different number of selecting trials k in Airplane case

Real-World Regression Problems

Problems	EI-ELM (50 Sigmoid hidden nodes)				I-ELM (500 Sigmoid hidden nodes, $k = 1$)	
	$k = 10$		$k = 20$			
	Mean	Dev	Mean	Dev	Mean	Dev
Abalone	0.0878	0.0033	0.0876	0.0015	0.0876	0.0033
Ailerons	0.0640	0.0066	**0.0571**	0.0022	0.0824	0.0232
Airplane	0.0922	0.0061	0.0862	0.0040	0.0898	0.0067
Auto Price	0.0924	0.0112	**0.0897**	0.0104	0.0948	0.0158
Bank	0.1066	0.0058	0.0896	0.0036	**0.0757**	0.0032
Boston	0.1133	0.0101	0.1102	0.0061	0.1084	0.0096
California	0.1591	0.0034	0.1548	0.0033	0.1543	0.0019
Census (8L)	0.0899	0.0017	0.0865	0.0011	0.0871	0.0018
Computer Activity	0.1075	0.0057	**0.0991**	0.0036	0.1057	0.0078
Delta Ailerons	0.0474	0.0062	0.0467	0.0042	0.0468	0.0052
Delta Elevators	0.0615	0.0049	**0.0586**	0.0038	0.0640	0.0055
Kinematics	0.1420	0.0029	0.1416	0.0019	0.1406	0.0014
Machine CPU	0.0498	0.0155	0.0467	0.0148	0.0474	0.0040
Puma	0.1846	0.0018	0.1827	0.0017	0.1856	0.0039
Pyrim	0.1514	0.0419	**0.1300**	0.0405	0.1712	0.0626
Servo	0.1634	0.0129	0.1558	0.0121	0.1589	0.0124

Table 7: Performance comparison between EI-ELM with 50 Sigmoid hidden nodes and I-ELM with 500 Sigmoid hidden nodes

Real-World Regression Problems

Problems	EI-ELM (50 RBF hidden nodes)				I-ELM (500 RBF hidden nodes, $k = 1$)	
	$k = 10$		$k = 20$			
	Mean	Dev	Mean	Dev	Mean	Dev
Abalone	0.0907	0.0034	0.0871	0.0023	0.0872	0.0022
Ailerons	0.0973	0.0229	**0.0775**	0.0033	0.1129	0.0295
Airplane	0.0943	0.0168	0.0813	0.0102	0.0772	0.0082
Auto Price	0.1187	0.0159	**0.1104**	0.0148	0.1231	0.0133
Bank	0.0989	0.0031	0.0888	0.0023	0.0843	0.0058
Boston	0.1197	0.0107	**0.1171**	0.0078	0.1214	0.0103
California	0.1624	0.0049	0.1579	0.0027	0.1582	0.0027
Census (8L)	0.0864	0.0026	0.0846	0.0020	0.0860	0.0018
Computer Activity	0.1295	0.0068	**0.1201**	0.0024	0.1358	0.0177
Delta Ailerons	0.0469	0.0067	**0.0466**	0.0039	0.0544	0.0076
Delta Elevators	0.0603	0.0049	**0.0602**	0.0039	0.0685	0.0099
Kinematics	0.1346	0.0025	**0.1306**	0.0019	0.1425	0.0095
Machine CPU	0.0622	0.0281	**0.0511**	0.0114	0.0614	0.0274
Puma	0.1789	0.0020	**0.1770**	0.0012	0.1850	0.0119
Pyrim	0.1214	0.0345	**0.0989**	0.0286	0.2179	0.1545
Servo	0.1487	0.0133	0.1434	0.0120	0.1410	0.0151

Table 8: Performance comparison between EI-ELM with 50 RBF hidden nodes and I-ELM with 500 RBF hidden nodes

Nature of Sequential Learning

Natural Learning

1. The training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm/system.
2. At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned.
3. A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed.
4. The learning algorithm/system has no prior knowledge as to how many training observations will be presented.

G.-B. Huang, et al., "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Nature of Sequential Learning

Natural Learning

1. The training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm/system.
2. At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned.
3. A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed.
4. The learning algorithm/system has no prior knowledge as to how many training observations will be presented.

G.-B. Huang, et al., "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Nature of Sequential Learning

Natural Learning

1. The training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm/system.
2. At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned.
3. A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed.
4. The learning algorithm/system has no prior knowledge as to how many training observations will be presented.

G.-B. Huang, et al., "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Nature of Sequential Learning

Natural Learning

1. The training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm/system.
2. At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned.
3. A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed.
4. The learning algorithm/system has no prior knowledge as to how many training observations will be presented.

G.-B. Huang, et al., "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Nature of Sequential Learning

Natural Learning

1. The training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm/system.
2. At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned.
3. A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed.
4. The learning algorithm/system has no prior knowledge as to how many training observations will be presented.

G.-B. Huang, et al., "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Popular Sequential Learning Methods

RAN Based

1. RAN, MRAN, GAP-RBF, GGAP-RBF
2. At any time, only the newly arrived single observation is seen and learned
3. They do not handle chunks of training observations
4. Many control parameters need to be fixed by human. Very laborious! Very tedious!
5. Training time is usually huge!!
6. Many control parameters need to be fixed by human

BP Based

1. Stochastic gradient BP (SGBP)
2. It may handle chunks of training observations

Online Sequential ELM (OS-ELM)

Two-Step Learning Model

1. Initialization phase: where batch ELM is used to initialize the learning system.
2. Sequential learning phase: where recursive least square (RLS) method is adopted to update the learning system sequentially.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Online Sequential ELM (OS-ELM)

Two-Step Learning Model

1. Initialization phase: where batch ELM is used to initialize the learning system.
2. Sequential learning phase: where recursive least square (RLS) method is adopted to update the learning system sequentially.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Online Sequential ELM (OS-ELM)

Two-Step Learning Model

1. Initialization phase: where batch ELM is used to initialize the learning system.
2. Sequential learning phase: where recursive least square (RLS) method is adopted to update the learning system sequentially.

N.-Y. Liang, et al., "A fast and accurate on-line sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411-1423, 2006.

Online Sequential ELM (OS-ELM)

$$\left\| \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} \beta - \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \right\| \tag{4}$$

$$\begin{aligned} \beta^{(1)} &= \mathbf{K}_1^{-1} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \\ &= \mathbf{K}_1^{-1}(\mathbf{K}_1\beta^{(0)} - \mathbf{H}_1^T\mathbf{H}_1\beta^{(0)} + \mathbf{H}_1^T\mathbf{T}_1) \\ &= \beta^{(0)} + \mathbf{K}_1^{-1}\mathbf{H}_1^T(\mathbf{T}_1 - \mathbf{H}_1\beta^{(0)}) \end{aligned} \tag{5}$$

where $\beta^{(1)}$ is the output weight for all the data learned so far,

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} = \mathbf{K}_0 + \mathbf{H}_1^T\mathbf{H}_1, \ \mathbf{K}_0 = \mathbf{H}_0^T\mathbf{H}_0, \ \beta^{(0)} = \mathbf{K}_0^{-1}\mathbf{H}_0^T\mathbf{T}_0 \tag{6}$$

Real-World Regression Problems

Datasets	Algorithms	Time (seconds)	RMSE		# nodes
			Training	Testing	
Auto-MPG	**OS-ELM (Sigmoid)**	0.0444	0.0680	0.0745	25
	OS-ELM (RBF)	0.0915	0.0696	0.0759	25
	Stochastic BP	0.0875	0.1112	0.1028	13
	GAP-RBF	0.4520	0.1144	0.1404	3.12
	MRAN	1.4644	0.1086	0.1376	4.46
Abalone	**OS-ELM (Sigmoid)**	0.5900	0.0754	0.0777	25
	OS-ELM (RBF)	1.2478	0.0759	0.0783	25
	Stochastic BP	0.7472	0.0996	0.0972	11
	GAP-RBF	83.784	0.0963	0.0966	23.62
	MRAN	1500.4	0.0836	0.0837	87.571
California Housing	**OS-ELM (Sigmoid)**	3.5753	0.1303	0.1332	50
	OS-ELM (RBF)	6.9629	0.1321	0.1341	50
	Stochastic BP	1.6866	0.1688	0.1704	9
	GGAP-RBF	115.34	0.1417	0.1386	18
	MRAN	2891.5	0.1598	0.1586	64

Table 9: Comparison between OS-ELM and other sequential algorithms on regression applications.

Real-World Classification Problems

Datasets	Algorithms	Time (seconds)	Accuracy (%)		# nodes
			Training	Testing	
Image Segmentation	**OS-ELM (Sigmoid)**	9.9981	97.00	94.88	180
	OS-ELM (RBF)	12.197	96.65	94.53	180
	Stochastic BP	2.5776	83.71	82.55	80
	GAP-RBF	1724.3	-	89.93	44.2
	MRAN	7004.5	-	93.30	53.1
Satellite Image	**OS-ELM (Sigmoid)**	302.48	91.88	88.93	400
	OS-ELM (RBF)	319.14	93.18	89.01	400
	Stochastic BP	3.1415	85.23	83.75	25
	MRAN	2469.4	-	86.36	20.4
DNA	**OS-ELM (Sigmoid)**	16.742	95.79	93.43	200
	OS-ELM (RBF)	20.951	96.12	94.37	200
	Stochastic BP	1.0840	85.64	82.11	12
	MRAN	6079.0	-	86.85	5

Table 10: Comparison between OS-ELM and other sequential algorithms on classification applications.

Time-Series Problems

Algorithms	Time (seconds)	Training RMSE	Testing RMSE	#nodes
OS-ELM (Sigmoid)	7.1148	0.0177	0.0183	120
OS-ELM (RBF)	10.0603	0.0184	0.0186	120
GGAP-RBF	24.326	0.0700	0.0368	13
MRAN	57.205	0.1101	0.0337	16
RANEKF	62.674	0.0726	0.0240	23
RAN	58.127	0.1006	0.0466	39

Table 11: Comparison between OS-ELM and other sequential algorithms on Mackey-Glass time series application.

Real-World Regression Problems

Datasets	Activation Functions	Algorithms	Learning Mode	Time (seconds)	RMSE		# nodes
					Training	Testing	
Auto - MPG	Sigmoid	ELM	Batch	0.0053	0.0697	0.0694	25
		OS-ELM	1-by-1	0.0444	0.0680	0.0745	25
			20-by-20	0.0150	0.0684	0.0738	25
			[10,30]	0.0213	0.0680	0.0765	25
	RBF	ELM	Batch	0.0100	0.0691	0.0694	25
		OS-ELM	1-by-1	0.0915	0.0696	0.0759	25
			20-by-20	0.0213	0.0686	0.0769	25
			[10,30]	0.0250	0.0692	0.0746	25
California Housing	Sigmoid	ELM	Batch	0.5122	0.1306	0.1333	50
		OS-ELM	1-by-1	3.5753	0.1303	0.1332	50
			20-by-20	0.6500	0.1297	0.1333	50
			[10,30]	0.8338	0.1302	0.1327	50
	RBF	ELM	Batch	1.0210	0.1292	0.1312	50
		OS-ELM	1-by-1	6.9629	0.1321	0.1341	50
			20-by-20	0.9794	0.1312	0.1333	50
			[10,30]	1.3241	0.1305	0.1326	50

Table 12: Performance comparison of ELM and OS-ELM on regression applications.

Real-World Classification Problems

Datasets	Activation Functions	Algorithms	Learning Mode	Time (seconds)	Accuracy (%)		# nodes
					Training	Testing	
Image Segmentation	Sigmoid	ELM	Batch	0.6384	96.75	95.07	180
		OS-ELM	1-by-1	9.9981	97.00	94.88	180
			20-by-20	1.0922	97.05	94.60	180
			[10,30]	0.9881	97.00	94.92	180
	RBF	ELM	Batch	1.6300	96.22	94.91	180
		OS-ELM	1-by-1	12.197	96.65	94.53	180
			20-by-20	1.4275	96.70	94.55	180
			[10,30]	1.4456	96.75	94.60	180
DNA	Sigmoid	ELM	Batch	0.9748	96.90	94.30	200
		OS-ELM	1-by-1	16.743	95.79	93.43	200
			20-by-20	1.7322	95.87	93.46	200
			[10,30]	1.7875	95.81	93.42	200
	RBF	ELM	Batch	8.2998	95.87	92.33	200
		OS-ELM	1-by-1	20.951	96.12	94.37	200
			20-by-20	2.6538	96.19	94.30	200
			[10,30]	2.8814	96.17	94.25	200

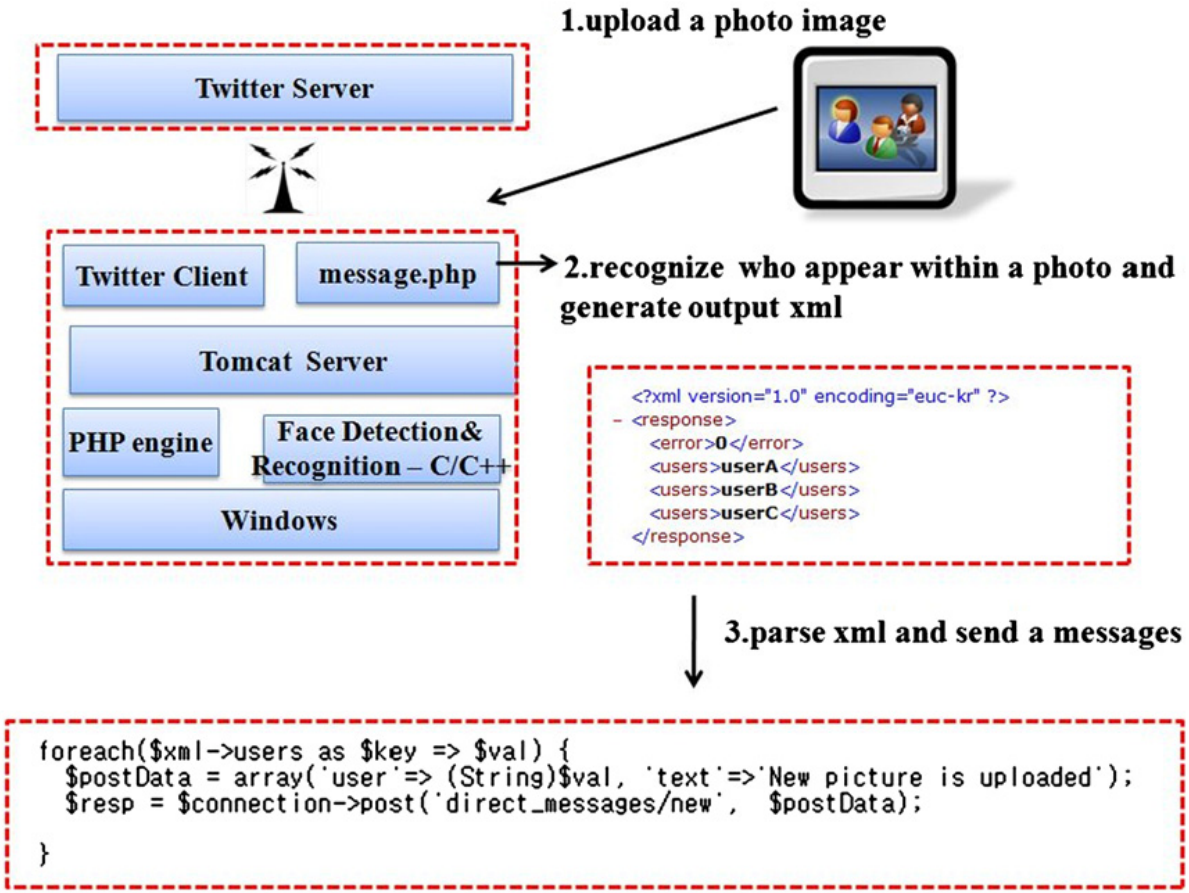
Table 13: Performance comparison of ELM and OS-ELM on classification applications.

Time-Series Problems

Activation Functions	Algorithms	Learning Mode	Time (seconds)	RMSE		# nodes
				Training	Testing	
Sigmoid	ELM	Batch	1.1664	0.0183	0.0187	120
	OS-ELM	1-by-1	7.1184	0.0177	0.0183	120
		20-by-20	0.9894	0.0177	0.0183	120
		[10,30]	1.0440	0.0185	0.0190	120
RBF	ELM	Batch	2.1794	0.0185	0.0180	120
	OS-ELM	1-by-1	10.060	0.0184	0.0186	120
		20-by-20	1.5574	0.0183	0.0186	120
		[10,30]	1.7441	0.0184	0.0187	120

Table 14: Performance comparison of ELM and OS-ELM on Mackey-Glass time series application.

Intelligent Photo Notification System For Twitter Service



K. Choi, et al., "Incremental face recognition for large-scale social network services", *Pattern Recognition*, vol. 45, pp. 2868-2883, 2012.

Intelligent Photo Notification System For Twitter Service

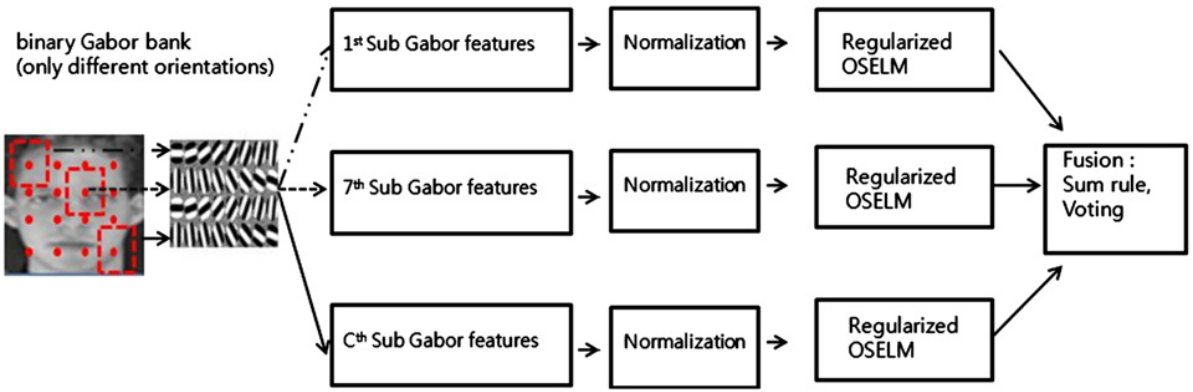


Figure 9: Binary Gabor filter-based OS-ELM (BG-OSELM)

Methods	Baseline		Sequential Subspace			Sequential Classifiers		
Database	PCA	FDA	CCIPCA	IPCA	ILDA	OSELM	BG-OSELM(S)	BG-OSELM(V)
AR	77.0	72.3	55.0	77.3	76.6	80.3	92.0	87.6
EYALE	99.7	96.9	58.5	99.7	100.0	100.0	99.7	99.7
BIOID	98.1	97.3	91.6	97.5	-	98.5	97.4	96.7
ETRI	95.8	95.5	86.9	95.4	-	97.2	97.0	94.2

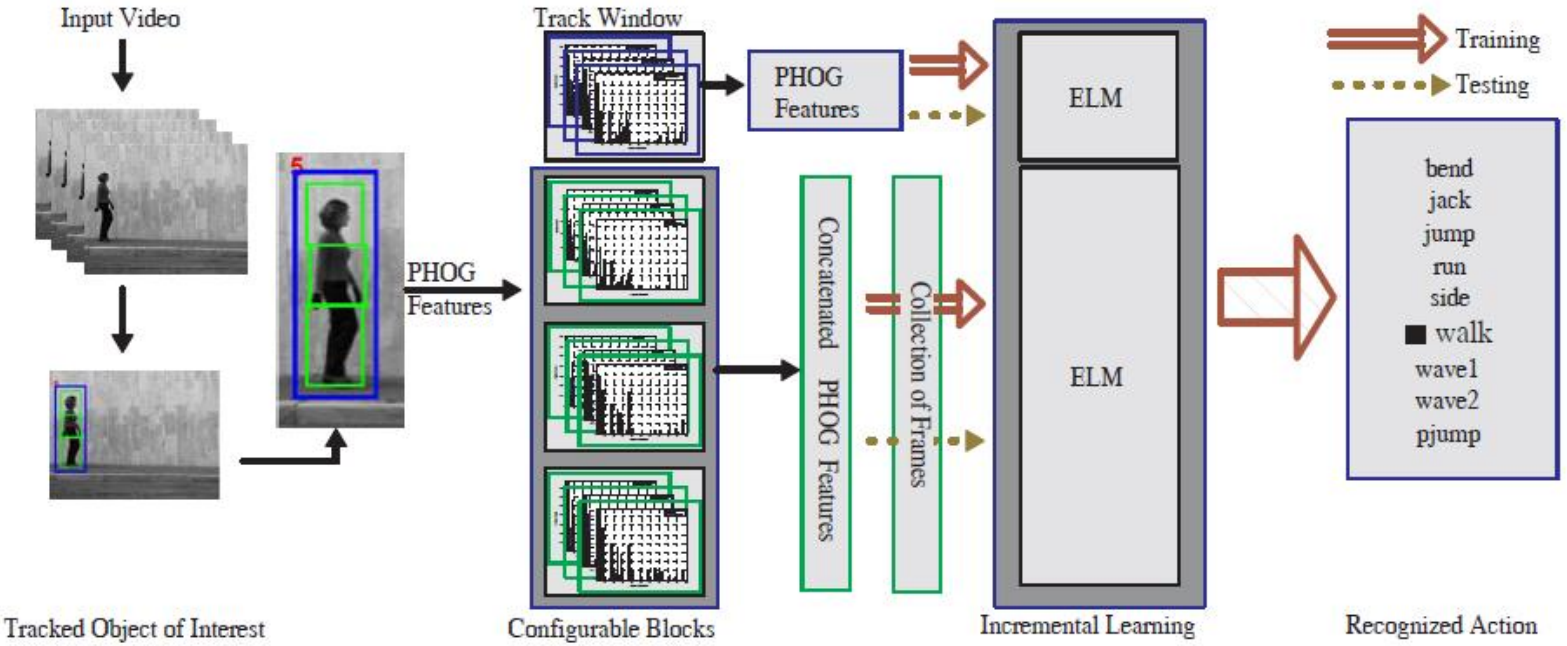
Table 15: Performance comparison of different sequential methods.

Online Sequential Human Action Recognition

Figure 10: Example frames from top row: Weizmann dataset, middle row: KTH dataset, and bottom row: UCF sports dataset

R. Minhas, et al., "Incremental learning in human action recognition based on *Snippets*", *(in press) IEEE Transactions on Circuits and Systems for Video Technology*, 2012.

Online Sequential Human Action Recognition



R. Minhas, et al., "Incremental learning in human action recognition based on *Snippets*", *(in press) IEEE Transactions on Circuits and Systems for Video Technology*, 2012.

Online Sequential Human Action Recognition

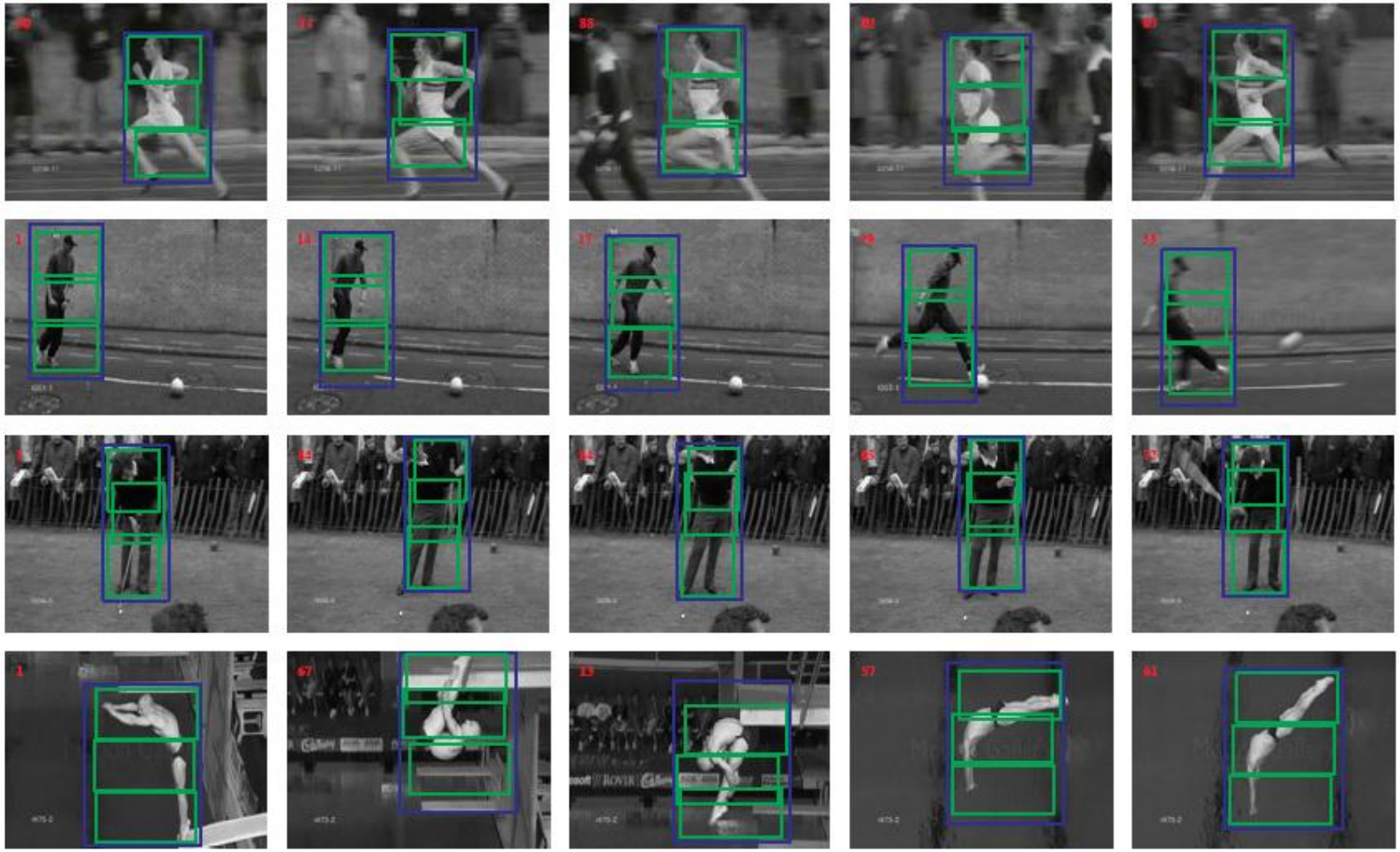


Figure 11: Tracking results using action videos of *run*, *kick*, *golf* and *dive* (top to bottom) from UCF Sports dataset

Weizmann dataset											
Methods	OS-ELM Based				[32]	[14]	[36]			[11]	
Frames	1/1	3/3	6/6	10/10	1/12	1/9	1/1	7/7	10/10	8/8	20/20
Accuracy	65.2	95.0	99.63	99.9	55.0	93.8	93.5	96.6	99.6	97.05	98.68

KTH dataset											
Methods	OS-ELM Based				[25]	[33]	[43]	[14]	[36]		[12]
Frames	1/1	3/3	6/6	10/10	-	-	-	-	1/1	7/7	20/20
Accuracy	74.4	88.5	92.5	94.4	91.3	90.3	83.9	91.7	88.0	90.9	90.84

Table 16: Classification comparison against different approaches at *snippet*-level.

Weizmann dataset											
Methods	OS-ELM Based				[2]	[32]	[14]	[36]	[41]	[30]	[11]
Frames	1/1	3/3	6/6	10/10	-	-	-	-	-	-	-
Accuracy	100.0	100.0	100.0	100.0	100.0	72.8	98.8	100.0	97.8	99.44	100.0

KTH dataset											
Methods	OS-ELM Based				[14]	[36]	[30]	[21]	[27]	[9]	[44]
Frames	1/1	3/3	6/6	10/10	-	-	-	-	-	-	-
Accuracy	92.8	93.5	95.7	96.1	91.7	92.7	94.83	95.77	97.0	96.7	95.7

Table 17: Classification comparison against different approaches at *sequence*-level.

R. Minhas, et al., "Incremental learning in human action recognition based on *Snippets*", *(in press) IEEE Transactions on Circuits and Systems for Video Technology*, 2012.

Open Problems

1. As observed in experimental studies, the performance of basic ELM is stable in a wide range of number of hidden nodes. Compared to the BP learning algorithm, the performance of basic ELM is not very sensitive to the number of hidden nodes. However, how to prove it in theory remains open.
2. One of the typical implementations of ELM is to use random nodes in the hidden layer and the hidden layer of SLFNs need not be tuned. It is interesting to see that the generalization performance of ELM turns out to be very stable. How to estimate the oscillation bound of the generalization performance of ELM remains open too.
3. It seems that ELM performs better than other conventional learning algorithms in applications with higher noise. How to prove it in theory is not clear.
4. ELM always has faster learning speed than LS-SVM if the same kernel is used?

Open Problems

5. ELM provides a batch learning kernel solution which is much simpler than other kernel learning algorithms such as LS-SVM. It is known that it may not be straightforward to have an efficient online sequential implementation of SVM and LS-SVM. However, due to the simplicity of ELM, is it possible to implement the online sequential variant of the kernel based ELM?
6. ELM always provides similar or better generalization performance than SVM and LS-SVM if the same kernel is used (if not affected by computing devices' precision)?
7. ELM tends to achieve better performance than SVM and LS-SVM in multiclasses applications, the higher the number of classes is, the larger the difference of their generalization performance will be?
8. Scalability of ELM with kernels in super large applications.
9. Parallel and distributed computing of ELM.
10. ELM will make real-time reasoning feasible?