Non-Iterative Methods for Classification, Forecasting and Visual Tracking

Part I: Randomization Based Neural Networks & Kernel Ridge Regression

Dr P. N. Suganthan epnsugan@ntu.edu.sg
School of EEE, NTU, Singapore

Some Software Resources Available from:
https://github.com/P-N-Suganthan

DeepLearn 2019
Warsaw, Poland
22 July – 26 July 2019

SEMCCO 2020 & FANCCO 2020

(includes all neural, fuzzy, swarm and evolutionary topics)

HUST, Wuhan, China

June 2020

IEEE SSCI 2019, CIEL 2019, SDE 2019, SIS 2019, etc.

http://ssci2019.org/

(includes all neural, fuzzy, swarm and evolutionary topics)

Randomization-Based ANN, Pseudo-Inverse Based Solutions, Kernel Ridge Regression, Random Forest and Related Topics

http://www.ntu.edu.sg/home/epnsugan/index_files/RNN-Moore-Penrose.htm

http://www.ntu.edu.sg/home/epnsugan/index_files/publications.htm

NANYANG TECHNOLOGICAL UNIVERSITY

Non-iterative Learning Methods

- These methods include: Randomization based neural networks (e.g. RVFL), Kernel ridge regression (KRR), random forest (RF).

- Randomization based NN methods have closed form solutions. Hence, trained single hidden layer networks can be obtained in a single step. However, multi-layer randomized networks require iterative application of the closed-form solution procedure.

- KRR has a closed form solution which can be computed easily for small number of training samples. For large number of training samples, approximations are required.

- In the batch mode, RF is non-iterative because once a node/tree is grown, it is not adjusted. However, RF iterates through the training data. DT/RF is potentially the best for explainable / interpretable AI.

Non-iterative Learning Methods

- Randomization based single hidden layer neural networks are extremely fast, although their accuracy is not very high. Suitable for mobile devices, hardware implementation, online applications, etc.
- RVFL has good scalability, except when both the number of training samples and number of features are large.

- KRR is also fast to train (for small number of training samples) with competitive accuracy. Its scalability is affected by the number of training samples. N training samples requires inverting a matrix of size NxN.

- RF is usually slow to train on a sequential computer. But, most RF variants are easy to parallelize as each decision tree in a forest can be trained independently (bagging type). Different RF variants have differing complexity with boosting types are the slowest due to sequential training requirement. RFs are highly accurate for offline mode classification.

NANYANG TECHNOLOGICAL UNIVERSITY

Outline

Part I: Classification by

- Randomized Neural Networks like RVFL (to slide 28)
- Kernel Ridge Regression (KRR) (slides 29-50)
- Random Forest (RF) (slides 51- 149)

Part II: Time Series Forecasting (Slides 150- 245)

Time Series Classification (Slides 246-280)

(by RVFL, RF, Deep Learners and others)

Part III: Visual Tracking (Slides 281-310)

(by RVFL, RF, KRR, etc.)

This talk is primarily on predictions. Not on feature extraction.

NANYANG TECHNOLOGICAL UNIVERSITY

Neural Networks

- Theoretical proof about the universal approximation ability of standard multilayer feedforward network can be found in the reference below.

- Some conditions are:

  Arbitrary bounded/squashing functions.

  Availability of sufficiently many hidden units.

- Standard multilayer feedforward network can approximate virtually any function to any desired degree of accuracy.

Reference: Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." *Neural networks* 2.5 (1989): 359-366.

Weakness and Solution (from the late 1980s & 1990s)

- Some weakness of back-prop:

  Error surface usually have multiple local minima ; Slow convergence.

  Sensitivity to learning rate setting.

- An alternative:

  parameters in hidden layers can be randomly and appropriately generated and fixed (i.e. no learning).

  The parameters in the last layer can be computed by least squares.

  These neural nets are known as the Randomization Based Neural Nets

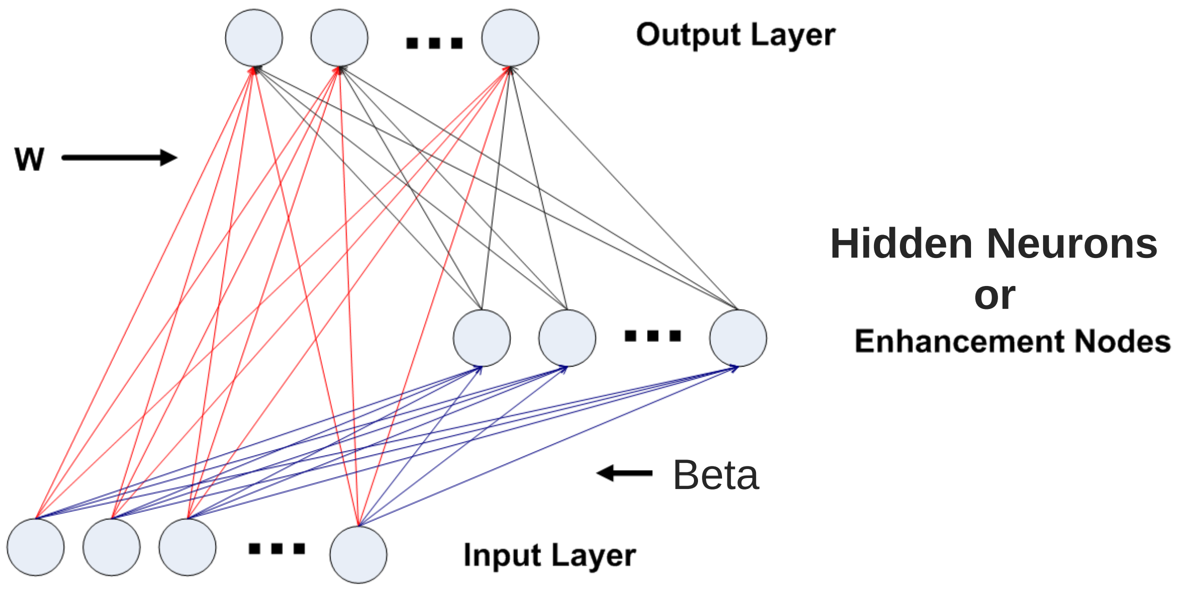
The following are the earliest independent publications, i.e. there is no citation of each other in all of these:

- **[Randomized SLFN]** Wouter F Schmidt, et al. "Feedforward neural networks with random weights". 11th IAPR Int Conf. on Pattern Recognition. **IEEE. 1992, pp. 1–4**
- **[RVFL]** Yoh-Han Pao, et al. "Learning and generalization characteristics of the random vector functional-link net". Neurocomputing **6.2 (1994), pp. 163–180.**
- **B. K. Verma and J. J. Mulawka,** "Training of the Multilayer Perceptron Using Direct Solution Methods," Proc. of the Twelfth IASTED Int Conf, Applied Informatics, pp. 18-24, May Annecy, France, 1994. [& "A modified backpropagation algorithm," pp. 840-844, IEEE WCCI 1994]
- **Braake et al.** "Random Activation Weight Neural Net (RAWN) for Fast Non-iterative Training", **Eng. Applns of AI, Elsevier, pp. 71-80, 1995.**
- **P. Guo, et al**, "An Exact Supervised Learning for a Three-Layer Supervised Neural Network", ICONIP'95, pp.1041--1044, Beijing, 1995**.. & "**A Vest of the Pseudoinverse Learning Algorithm" - arXiv , **2018; Researchgate, etc.**
- **A. Elissee, H. Paugam-Moisy**, JNN: A randomized algorithm for training multilayer networks in polynomial time, Neurocomputing 29 (1999) 3-24.
- **GB Huang, et al,** "Extreme learning machine: A new learning scheme of feedforward neural networks," Proc. of IEEE IJCNN, Vol. 2, 2004, pp. 985-990..

RVFL and ELM became popular with a number of follow-up publications.

Structure of RVFL (SLFN with direct links)

Weights Connecting → To output



Parameters β (indicated in blue) in enhancement (or hidden layer) nodes are randomly generated in a proper range and kept fixed.

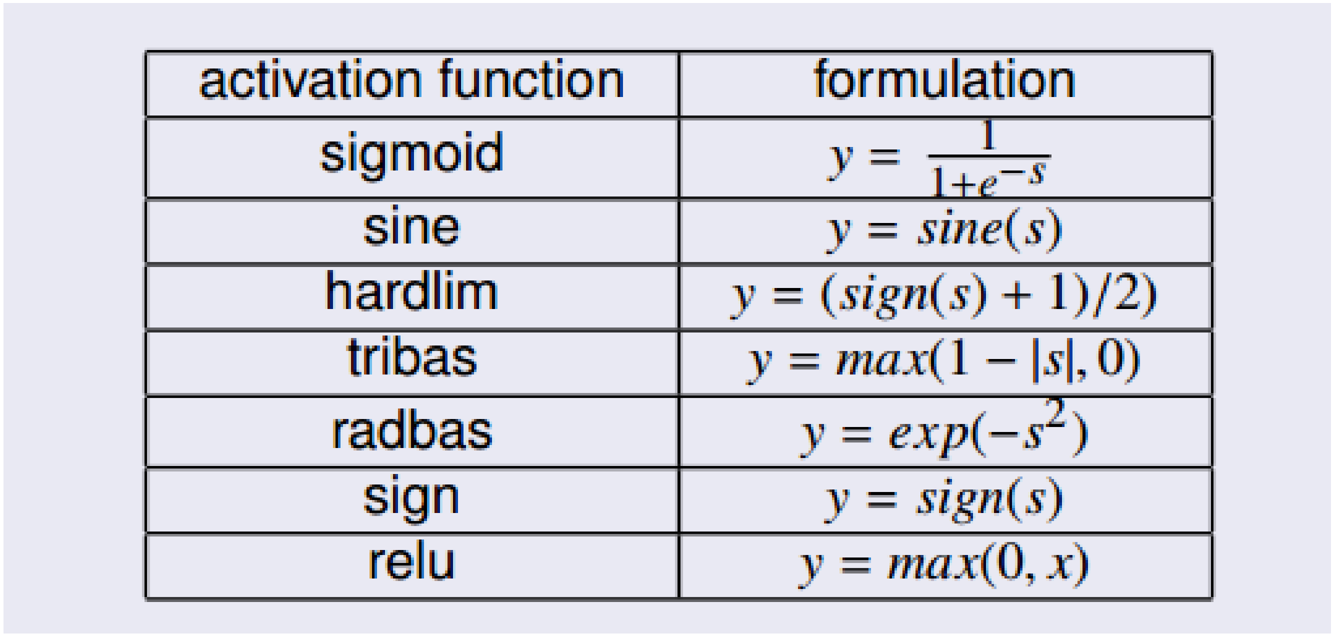
Original features (from the input layer) are concatenated with enhanced features (from the hidden layer) to boost the performance.

Learning aims at ideally obtaining $y_i = d_i * W$, $i = 1, 2, \ldots, n$, where n is the number of training sample, W (indicated in red and grey) are the weights in the output layer, and y, d represent the output and the combined features, respectively.

NANYANG TECHNOLOGICAL UNIVERSITY

RVFL details

- Notations:
- X = [x1, x2, ...xn]' : input data (n samples and m features).
- β = [β1, β2, ...βm]' : the weights for the enhancement nodes (m × k, k is the number of enhancement nodes).
- b = [b1, b2, ...bk] the bias for the enhancement node.
- H = h(X ∗ β + ones(n, 1) ∗ b): feature matrix after the enhancement nodes and h is the activation function.



activation function	formulation
sigmoid	$y = \frac{1}{1+e^{-s}}$
sine	$y = sine(s)$
hardlim	$y = (sign(s) + 1)/2)$
tribas	$y = max(1 - \|s\|, 0)$
radbas	$y = exp(-s^2)$
sign	$y = sign(s)$
relu	$y = max(0, x)$

Reference:
S. Scardapane and D. Wang, “Randomness in neural networks: an overview," WIREs Data Mining Knowledge Discovery, 2017.

RVFL details

- $$H = \begin{bmatrix} h(x_1) \\ h(x_2) \\ \vdots \\ h(x_n) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_k(x_1) \\ h_1(x_2) & \cdots & h_k(x_2) \\ \vdots & \ddots & \vdots \\ h_1(x_n) & \cdots & h_k(x_n) \end{bmatrix}$$

- $$D = [H, X] = \begin{bmatrix} h_1(x_1) & \cdots & h_k(x_1) & x_{11} & \cdots & x_{1m} \\ h_1(x_2) & \cdots & h_k(x_2) & x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ h_1(x_n) & \cdots & h_k(x_n) & x_{n1} & \cdots & x_{nm} \end{bmatrix} = \begin{bmatrix} d(x_1) \\ d(x_2) \\ \dots \\ \dots \\ \dots \\ d(x_n) \end{bmatrix}$$

- Target y, can be continuous value for regression problem and class labels for C class classification problem $(1, 2, ..., C)$.

- For classification, y is usually transformed into 0-1 coding of the class label. For example

$$y = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Learning in RVFL

- Once the random parameters β and b are generated in proper range, the learning of RVFL aims at solving the following problem: $o_i = d_i * W$, $i = 1, 2, ..., n$, where n is the number of training sample, W are the weights in the output layer, and o, d represent the output and the combined features.

- Objective: $\min_{W} ||DW-Y||^2 + \lambda ||W||^2$

- Solutions:

  In prime space: $W = (\lambda I + D^T D)^{-1} D^T Y$; (# training samples > total features)

  In dual space: $W = D^T (\lambda I + DD^T)^{-1} Y$ (total features > # training samples)

(Sometimes 1/C is used instead of λ)

where λ is the regularization parameter, when $\lambda \rightarrow 0$, it becomes the Moore-Penrose pseudoinverse. D and Y are the stacked features and target, respectively.

Regularization (or ridge regression or weight decay):

https://en.wikipedia.org/wiki/Tikhonov_regularization

Approximation Ability of RVFL

Approximation ability of RVFL:

Theoretical justification for RVFL is presented in [5]:

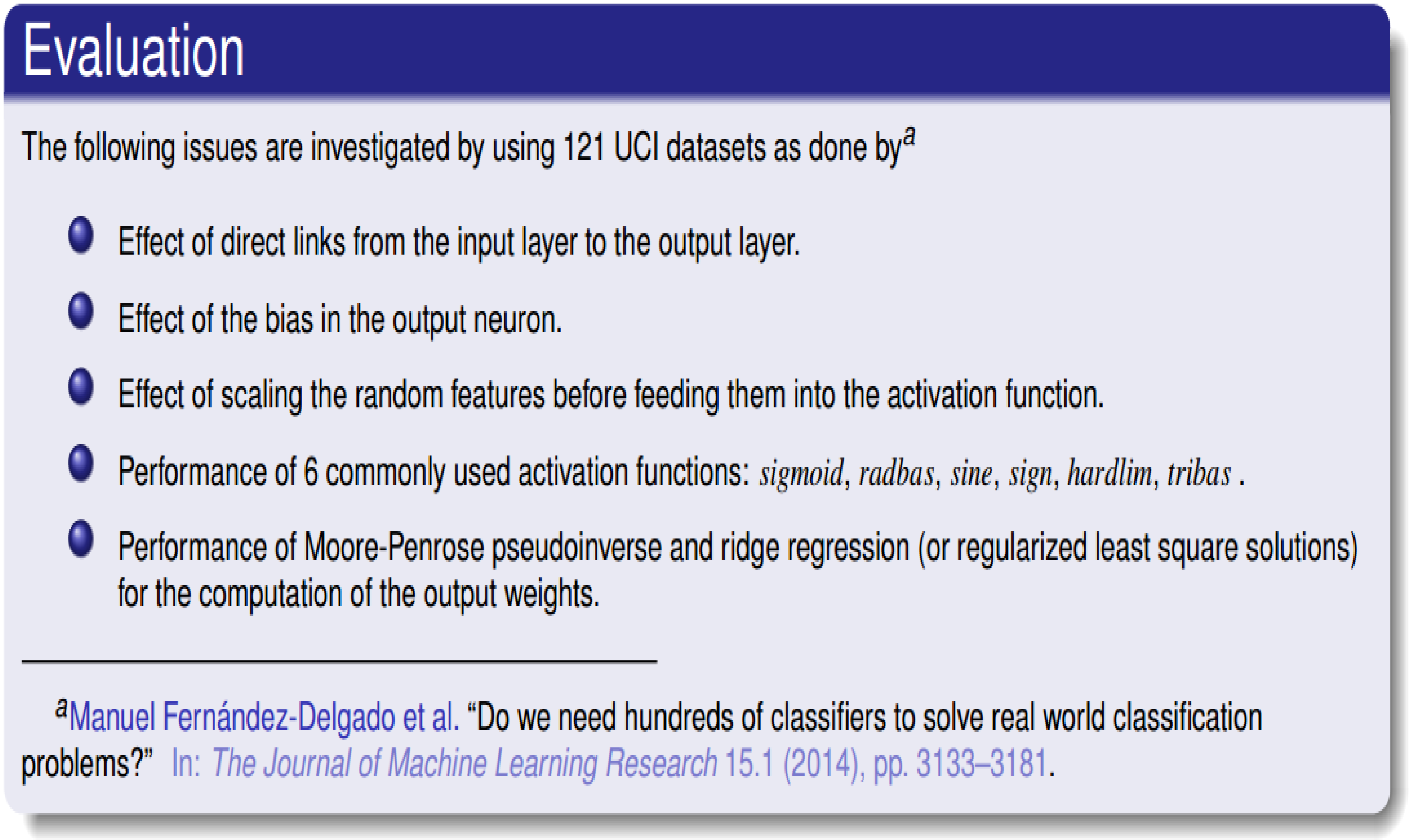
- Formulates a limit-integral representation of the function to be approximated
- Evaluates the integral with the Monte-Carlo Method

Conclusion:

- RVFL is a universal approximator for continuous function on bounded finite dimensional sets.
- RVFL is an efficient universal approximator with the rate of approximation error converge to zero in order *O(C/ sqrt(n))*, where *n* is the number of basis functions and *C* is independent of *n*.

[5] Bons Igelnik and Yoh-Han Pao. “Stochastic choice of basis functions in adaptive function approximation and the functional-link net”. In: IEEE Transactions on Neural Networks 6.6 (1995), pp. 1320–1329.

Comprehensive Comparisons of RVFL & Randomized Neural Networks



Evaluation

The following issues are investigated by using 121 UCI datasets as done by[a]

- Effect of direct links from the input layer to the output layer.
- Effect of the bias in the output neuron.
- Effect of scaling the random features before feeding them into the activation function.
- Performance of 6 commonly used activation functions: *sigmoid*, *radbas*, *sine*, *sign*, *hardlim*, *tribas* .
- Performance of Moore-Penrose pseudoinverse and ridge regression (or regularized least square solutions) for the computation of the output weights.

[a] Manuel Fernández-Delgado et al. "Do we need hundreds of classifiers to solve real world classification problems?" In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3133–3181.

EVALUATION PROTOCOL

Evaluation Metric: Friedman Ranking [6]

- Rank the algorithms for each dataset separately (the best performing algorithm getting the rank of 1, the second best rank 2 and so on). In case of ties, average ranks are assigned.
- Let *rji* be the rank of the *jth* of *k* algorithms on the *ith* of *N* datasets. The Friedman test compares the average ranks of algorithms: $R_j = \sum_i r_i^j$
- When *N* and *k* are large, we get the Friedman statistic $\chi_F{}^2 = \frac{12N}{k(k+1)}\left[\sum_j R^2{}_j - \frac{k(k+1)^2}{4}\right]$
- Then $F_F = \frac{(N-1)\chi_F{}^2}{N(k-1)-\chi_F{}^2}$ is distributed according to the F-distribution with *k − 1* and *(k − 1)(N − 1)* degrees of freedom

[6] Manuel Fernández-Delgado et al. “Do we need hundreds of classifiers to solve real world classification problems?” In: The Journal of Machine Learning Research 15.1 (2014), pp. 3133–3181.

Evaluation Protocol *(Cont.)*

- If the corresponding average ranks of two different algorithms differ by at least the critical difference $CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$, the performance of them are considered as significantly different.
- critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$. α is the significance level and is set to be 0.05 in this work

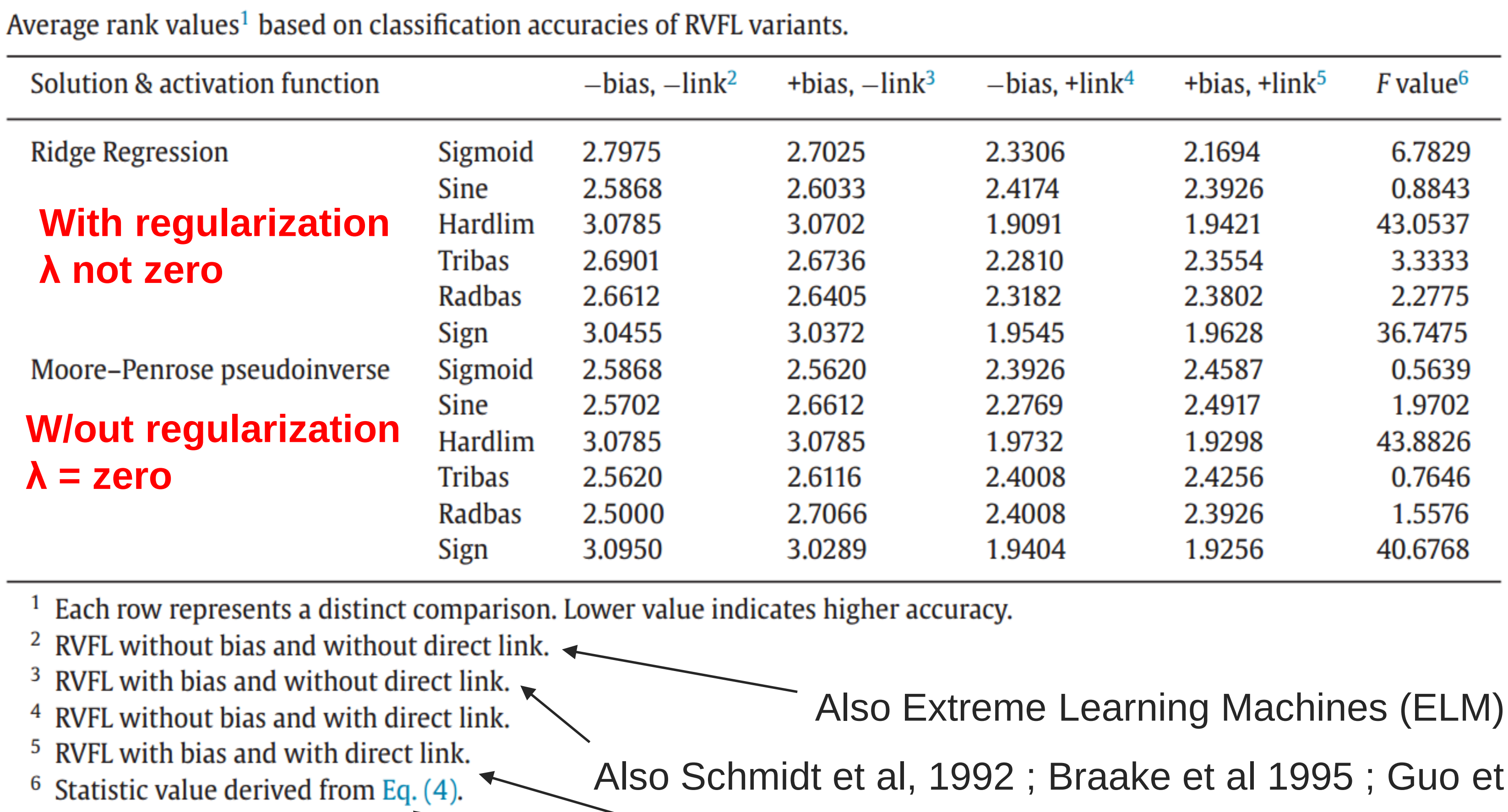
First Works to Compare Randomized Neural Networks:

L. Zhang, P. N. Suganthan, "A Comprehensive Evaluation of Random Vector Functional Link Networks," Information Sciences, **DOI:** 10.1016/j.ins.2015.09.025, Volumes 367–368, pp. 1094–1105, Nov 2016

Y. Ren, P. N. Suganthan, N. Srikanth, G. Amaratunga, "Random Vector Functional Link Network for Short-term Electricity Load Demand Forecasting", Information Sciences, Volumes 367–368, pp 1078–1093, Nov. 2016.

L. Zhang, P. N. Suganthan, "A Survey of Randomized Algorithms for Training Neural Networks," Information Sciences, DoI: 10.1016/j.ins.2016.01.039, Volumes 364–365, pp.146–155, Oct, 2016.

Results: direct link and bias



Average rank values[1] based on classification accuracies of RVFL variants.

Solution & activation function		−bias, −link[2]	+bias, −link[3]	−bias, +link[4]	+bias, +link[5]	*F* value[6]
Ridge Regression	Sigmoid	2.7975	2.7025	2.3306	2.1694	6.7829
	Sine	2.5868	2.6033	2.4174	2.3926	0.8843
	Hardlim	3.0785	3.0702	1.9091	1.9421	43.0537
	Tribas	2.6901	2.6736	2.2810	2.3554	3.3333
	Radbas	2.6612	2.6405	2.3182	2.3802	2.2775
	Sign	3.0455	3.0372	1.9545	1.9628	36.7475
Moore–Penrose pseudoinverse	Sigmoid	2.5868	2.5620	2.3926	2.4587	0.5639
	Sine	2.5702	2.6612	2.2769	2.4917	1.9702
	Hardlim	3.0785	3.0785	1.9732	1.9298	43.8826
	Tribas	2.5620	2.6116	2.4008	2.4256	0.7646
	Radbas	2.5000	2.7066	2.4008	2.3926	1.5576
	Sign	3.0950	3.0289	1.9404	1.9256	40.6768

With regularization
λ not zero

W/out regularization
λ = zero

[1] Each row represents a distinct comparison. Lower value indicates higher accuracy.
[2] RVFL without bias and without direct link.
[3] RVFL with bias and without direct link.
[4] RVFL without bias and with direct link.
[5] RVFL with bias and with direct link.
[6] Statistic value derived from Eq. (4).

Also Extreme Learning Machines (ELM), 2004

Also Schmidt et al, 1992 ; Braake et al 1995 ; Guo et al 1995

Also original RVFL 1992, 1994

Equation is 2 slides back

The direct links lead to better performance than those without in all cases.

The bias term in the output neurons only has mixed effects on the performance, as it may or may not improve performance.

RESULTS: ACTIVATION FUNCTION.

Statistical significance test for different activation functions. The values in bracket stands for their average rank. Lower values are for better performance. The mark √ indicates that these two methods are statistically significantly different.

(a) -bias, -link, Ridge regression

	sigmoid (3.18)	sine (2.67)	hardlim (4.83)	tribas (2.95)	radbas (2.48)	sign (4.89)
sigmoid (3.18)			√		√	√
sine (2.67)			√			√
hardlim (4.83)	√	√		√	√	
tribas (2.95)			√			√
radbas (2.48)	√		√			√
sign (4.89)	√	√		√	√	

(b) +bias, -link, Ridge regression

	sigmoid (3.16)	sine (2.69)	hardlim (4.80)	tribas (2.98)	radbas (2.49)	sign (4.88)
sigmoid (3.16)			√		√	√
sine (2.69)			√			√
hardlim (4.80)	√	√		√	√	
tribas (2.98)			√			√
radbas (2.49)	√		√			√
sign (4.88)	√	√		√	√	

(c) -bias, +link, Ridge regression

	sigmoid (3.45)	sine (2.91)	hardlim (4.36)	tribas (2.98)	radbas (2.76)	sign (4.55)
sigmoid (3.45)			√		√	√
sine (2.91)			√			√
hardlim (4.36)	√	√		√	√	
tribas (2.98)			√			√
radbas (2.76)	√		√			√
sign (4.55)	√	√		√	√	

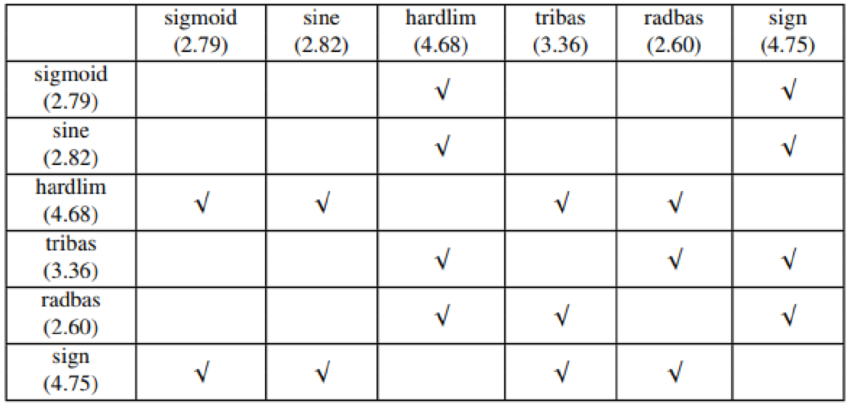
(d) +bias, +link, Ridge regression

	sigmoid (3.37)	sine (2.98)	hardlim (4.37)	tribas (3.02)	radbas (2.71)	sign (4.54)
sigmoid (3.37)			√		√	√
sine (2.98)			√			√
hardlim (4.37)	√	√		√	√	
tribas (3.02)			√			√
radbas (2.71)	√		√			√
sign (4.54)	√	√		√	√	

NANYANG TECHNOLOGICAL UNIVERSITY

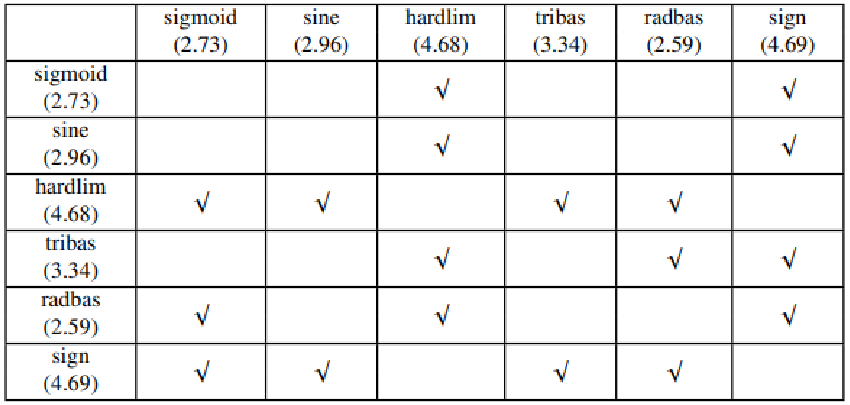
RESULTS: ACTIVATION FUNCTION.

(e) -bias, -link, Moore-Penrose pseudoinverse



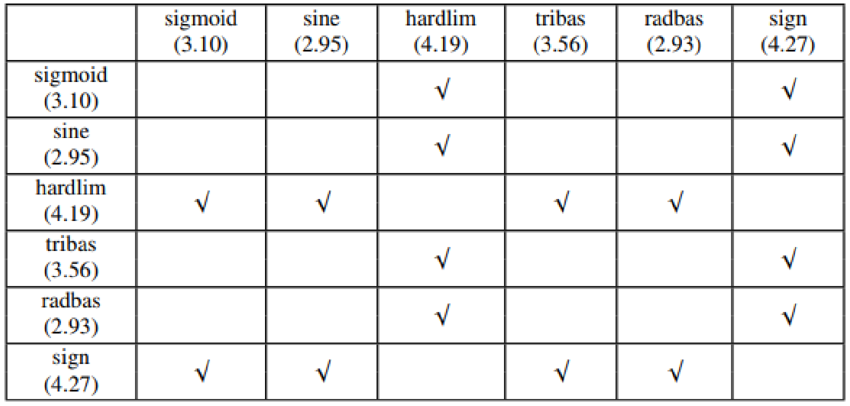
	sigmoid (2.79)	sine (2.82)	hardlim (4.68)	tribas (3.36)	radbas (2.60)	sign (4.75)
sigmoid (2.79)			√			√
sine (2.82)			√			√
hardlim (4.68)	√	√		√	√	
tribas (3.36)			√		√	√
radbas (2.60)			√	√		√
sign (4.75)	√	√		√	√	

(f) +bias, -link, Moore-Penrose pseudoinverse



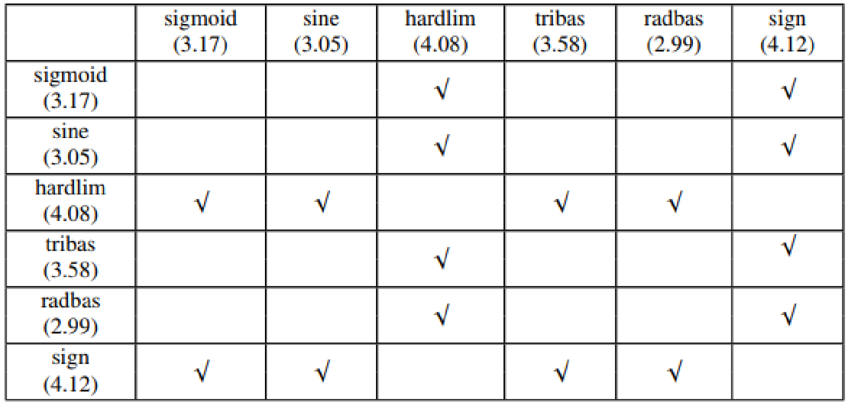
	sigmoid (2.73)	sine (2.96)	hardlim (4.68)	tribas (3.34)	radbas (2.59)	sign (4.69)
sigmoid (2.73)			√			√
sine (2.96)			√			√
hardlim (4.68)	√	√		√	√	
tribas (3.34)			√		√	√
radbas (2.59)	√		√			√
sign (4.69)	√	√		√	√	

(g) -bias,+link, Moore-Penrose pseudoinverse



	sigmoid (3.10)	sine (2.95)	hardlim (4.19)	tribas (3.56)	radbas (2.93)	sign (4.27)
sigmoid (3.10)			√			√
sine (2.95)			√			√
hardlim (4.19)	√	√		√	√	
tribas (3.56)			√			√
radbas (2.93)			√			√
sign (4.27)	√	√		√	√	

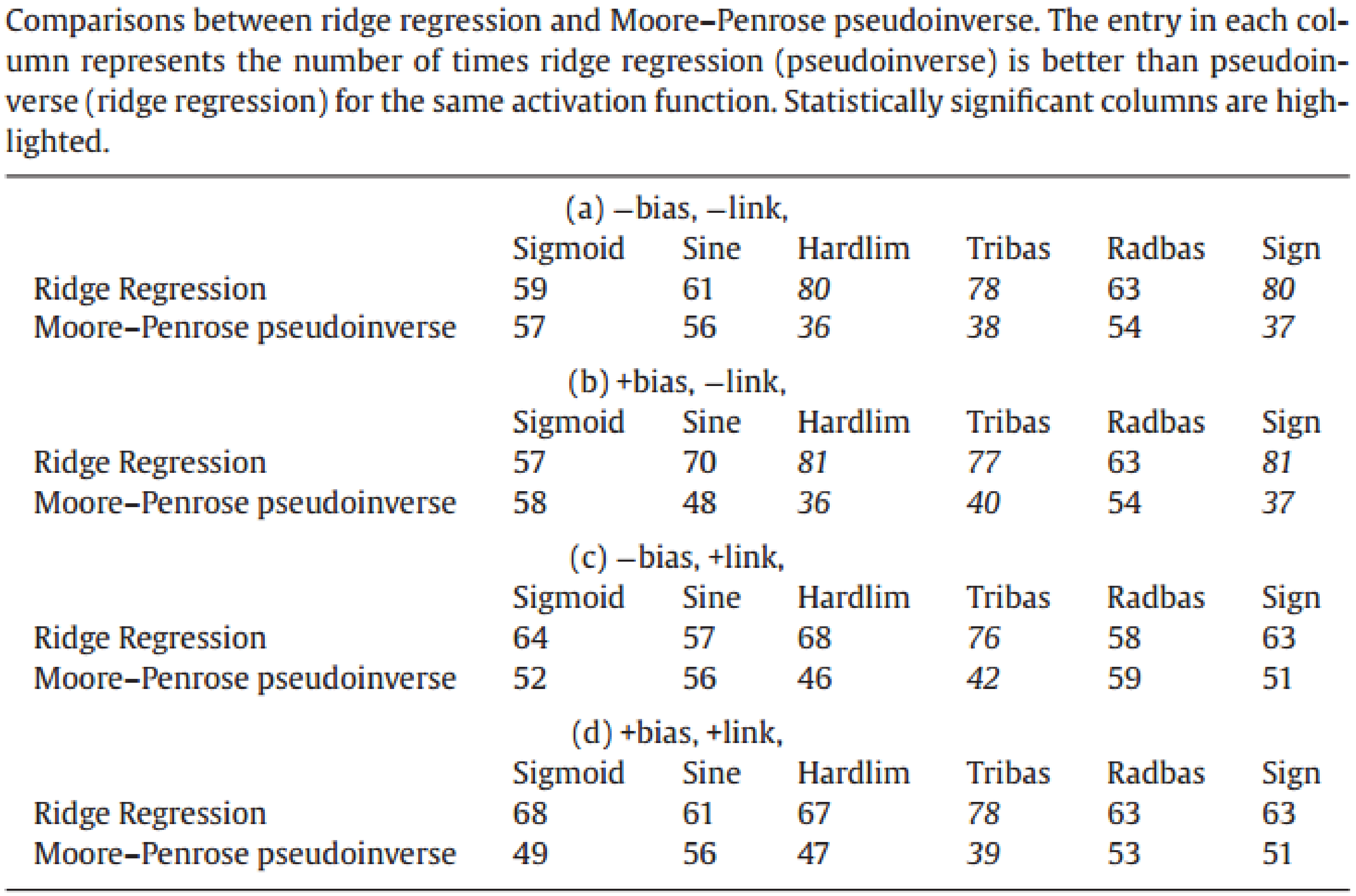
(h) +bias, +link, Moore-Penrose pseudoinverse



	sigmoid (3.17)	sine (3.05)	hardlim (4.08)	tribas (3.58)	radbas (2.99)	sign (4.12)
sigmoid (3.17)			√			√
sine (3.05)			√			√
hardlim (4.08)	√	√		√	√	
tribas (3.58)			√			√
radbas (2.99)			√			√
sign (4.12)	√	√		√	√	

Conclusion: $radbas > sine > tribas > sig > hardlim > sign$

Moore–Penrose Pseudoinverse (λ =0) VS Ridge Regression (λ not 0)



Comparisons between ridge regression and Moore–Penrose pseudoinverse. The entry in each column represents the number of times ridge regression (pseudoinverse) is better than pseudoinverse (ridge regression) for the same activation function. Statistically significant columns are highlighted.

(a) –bias, –link,

	Sigmoid	Sine	Hardlim	Tribas	Radbas	Sign
Ridge Regression	59	61	*80*	*78*	63	*80*
Moore–Penrose pseudoinverse	57	56	*36*	*38*	54	*37*

(b) +bias, –link,

	Sigmoid	Sine	Hardlim	Tribas	Radbas	Sign
Ridge Regression	57	70	*81*	*77*	63	*81*
Moore–Penrose pseudoinverse	58	48	*36*	*40*	54	*37*

(c) –bias, +link,

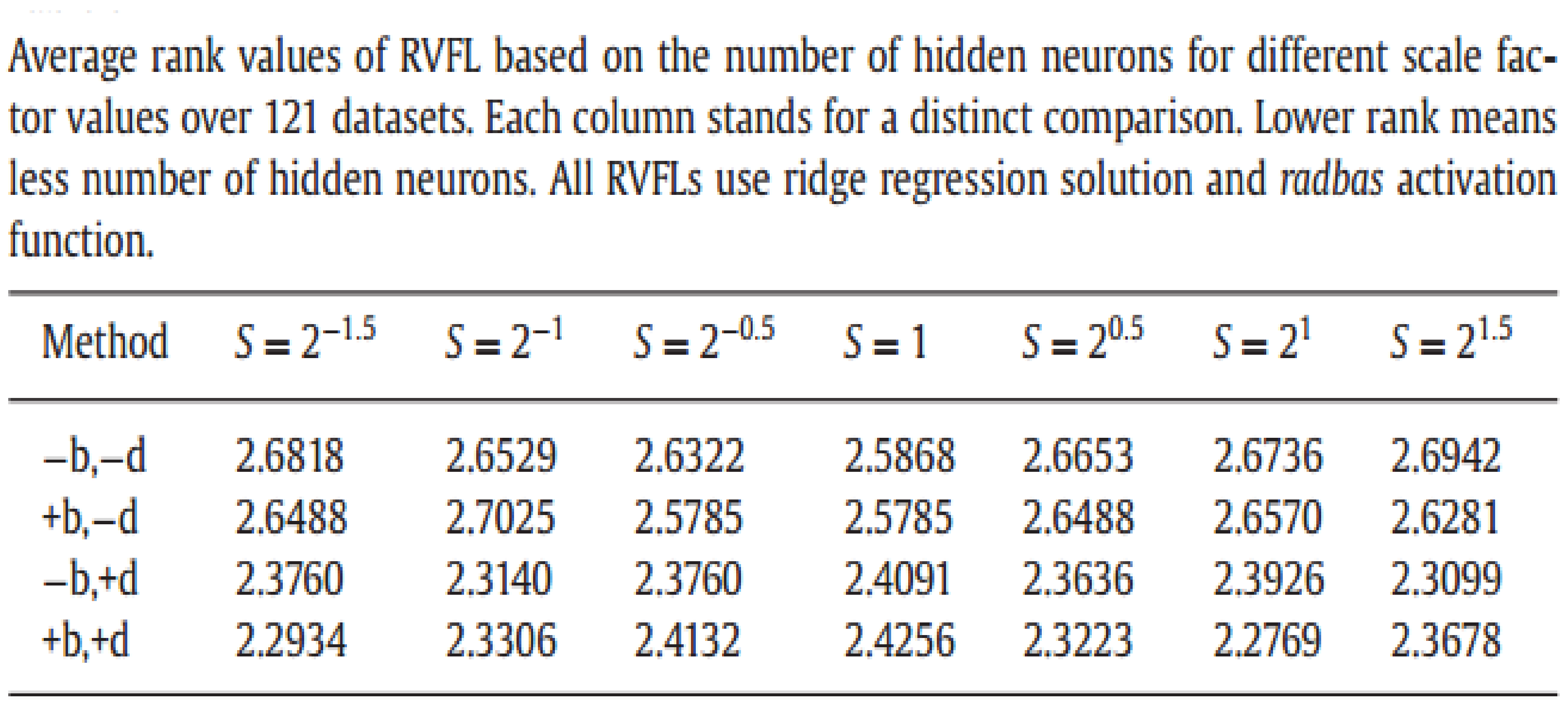
	Sigmoid	Sine	Hardlim	Tribas	Radbas	Sign
Ridge Regression	64	57	68	*76*	58	63
Moore–Penrose pseudoinverse	52	56	46	*42*	59	51

(d) +bias, +link,

	Sigmoid	Sine	Hardlim	Tribas	Radbas	Sign
Ridge Regression	68	61	67	*78*	63	63
Moore–Penrose pseudoinverse	49	56	47	*39*	53	51

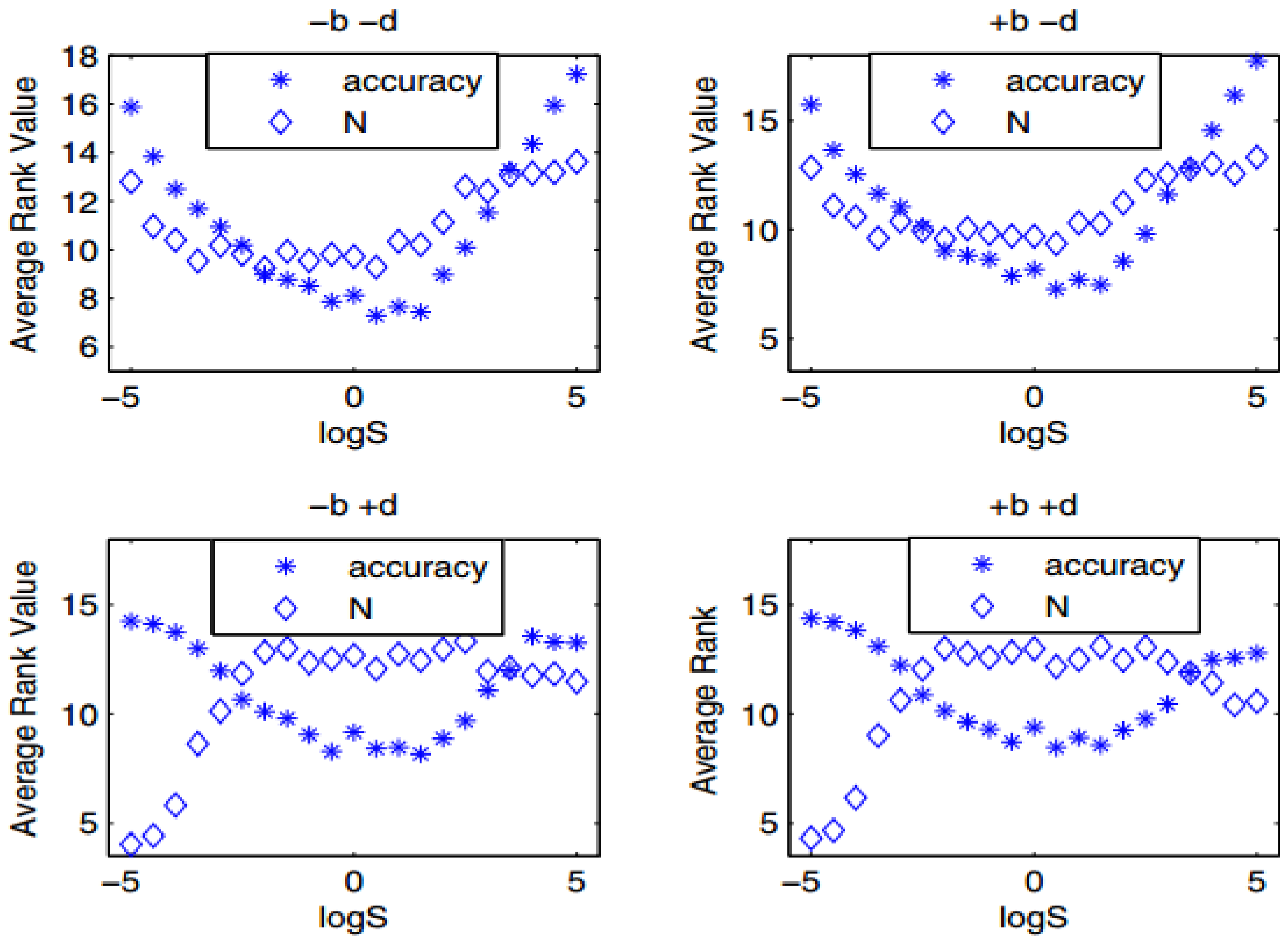
There are 4 independent comparisons.
Ridge regression based RVFL shows better performance than the Moore–Penrose pseudoinverse based RVFL

Results: Scaling the Randomization range of input weights and biases



Average rank values of RVFL based on the number of hidden neurons for different scale factor values over 121 datasets. Each column stands for a distinct comparison. Lower rank means less number of hidden neurons. All RVFLs use ridge regression solution and *radbas* activation function.

Method	$S = 2^{-1.5}$	$S = 2^{-1}$	$S = 2^{-0.5}$	$S = 1$	$S = 2^{0.5}$	$S = 2^{1}$	$S = 2^{1.5}$
−b,−d	2.6818	2.6529	2.6322	2.5868	2.6653	2.6736	2.6942
+b,−d	2.6488	2.7025	2.5785	2.5785	2.6488	2.6570	2.6281
−b,+d	2.3760	2.3140	2.3760	2.4091	2.3636	2.3926	2.3099
+b,+d	2.2934	2.3306	2.4132	2.4256	2.3223	2.2769	2.3678



Performance of RVFL based for different ranges of randomization. Smaller rank indicates better accuracy and less number of hidden neurons. N stands for the number of hidden neurons corresponding to the testing accuracy used in the ranking. In other words for each ranking performance is maximized and corresponding number of neurons recorded.

Results: Scaling the randomization range of input weights and biases

- Scaling down the randomization range of input weights and biases to avoid saturating the neurons may risk at degenerating the discrimination power of the random features. However, this can be compensated by having more hidden neurons.

- Scaling the randomization range of input weights and biases up to enhance the discrimination power of the random features may risk saturating the neurons. Again, this can be compensated by having more hidden neurons or combining with the direct link from the input to the output layer.

- However, we prefer lower model complexity, i.e. less number of hidden neurons.

NANYANG TECHNOLOGICAL UNIVERSITY

RVFL Vs Extreme Learning Machines

- The main difference between the RVFL developed in early 1990s and the ELM developed in mid 2000s is the direct links from inputs to the outputs (RVFL has while ELM doesn't). Hence, in the previous slides, RVFL without direct links is ELM (without bias too).
- Direct links from input to output increases the feature dimensionality in particular if the input feature dimensionality is large. Hence, RVFL may require a larger matrix inversion and longer training time.
- RVFL possibly uses non-linear features to deal with non-linear separation while linear separation is achieved by direct links from inputs to outputs. Hence, RVFL requires lesser non-linear hidden neurons, thereby requiring lower testing time.
- Linear links regularise the adverse effects of randomization.
- Experimental results in the context of classification consistently showed that the presence of direct links improve the accuracy of the RVFL.
- For Time series forecasting, with direct input-output links offer statistically superior performance over without. Direct links as time delayed line of FIR?
- Hence, permanently removing direct input-output links of RVFL in ELM is not a wise decision.

NANYANG TECHNOLOGICAL UNIVERSITY

24

Evidences of RVFL's Superiority Over ELM

Recently, there are several publications empirically showing overall superior performance of RVFL(1994) over ELM (2004):

- Y. Ren, P. N. Suganthan, N. Srikanth, and G. Amaratunga, "Random vector functional link network for short-term electricity load demand forecasting," Information Sciences, vol. 367, pp. 1078-1093, 2016.
- L. Zhang and P. Suganthan, "A comprehensive evaluation of random vector functional link networks," Information Sciences, vol. 367, pp. 1094-1105, 2016.
- L. Zhang and P. N. Suganthan, "Visual tracking with convolutional random vector functional link network," IEEE Transactions on Cybernetics, vol. 47, no. 10, pp. 3243-3253, 2017.
- N. Vukovic, M. Petrovic, and Z. Miljkovic, "A comprehensive experimental evaluation of orthogonal polynomial expanded random vector functional link neural networks for regression," Applied Soft Computing, September 2018.
- L. Tang, Y. Wu, and L. Yu, "A non-iterative decomposition-ensemble learning paradigm using RVFL network for crude oil price forecasting," Applied Soft Computing, September 2018.
- P. Henriquez and G. Ruz, "A non-iterative method for pruning hidden neurons in neural networks with random weights," Applied Soft Computing, September 2018.
- D. P. P. Mesquita, J. P. P. Gomes, L. R. Rodrigues, S. A. F. Oliveira, and R. K. H. Galvao, "Building selective ensembles of randomization based neural networks with the successive projections algorithm," Applied Soft Computing, September 2018.
- P. N. Suganthan, "On Non-iterative Learning Algorithms with Closed-form Solution", Applied Soft Computing, September 2018.
- Y. Zhang, J. Wu, Z. Cai, B. Du, P. S. Yu, An unsupervised parameter learning model for RVFL neural network, Neural Networks 112 (2019) 85-97.

doi:10.1016/j.neunet.**2019.01.007**

NANYANG TECHNOLOGICAL UNIVERSITY

Issues Raised in “A Vest of the Pseudoinverse Learning Algorithm”

- Researchers may wish to investigate issues raised in the above paper in Section 4 by Prof Ping Guo

“4. Incorrect Statements in ELM papers

In ELM papers, there also exists a lot of incorrect statements and false claims, here we list only some of them as follows:

4.1 Theorem 2.1 in [Huang2006]

The theorem 2.1 in [Huang2006] claimed that for the activation function g which is infinitely differentiable in any interval, with N training samples, randomly chosen input weights and hidden layer bias from any intervals, the hidden layer output matrix H is invertible. However, this theorem is incorrect.”

Similarly, Subsections 4.2 – 4.5 raise various issues…..

There is also an anonymous web page raising issues similar to The above …

Diversified RVFL Ensemble

- Motivation: RVFL can be an unstable classifier
- Train an ensemble of RVFL with different hyper-parameters by perturbation.
- Tune parameters for each dataset once.
- Generate 500 bags and train 500 RVFLs with randomly permutated parameters

Parameters	Tuned Result	Randomized Parameter
Hidden Neuron Numbers	$\mathcal{N}$	$Round((1+\delta) * \mathcal{N})$
Regularization Parameter	C	$(1+\delta) * C$
Scale Parameter	$\mathcal{S}$	$(1+\delta) * \mathcal{S}$

Reference: L. Zhang, P. N. Suganthan, Benchmarking Ensemble Classifiers with Novel Co-trained Kernel Ridge Regression and Random Vector Functional Link Ensembles, IEEE Computational Intelligence Magazine, Nov. 2017.

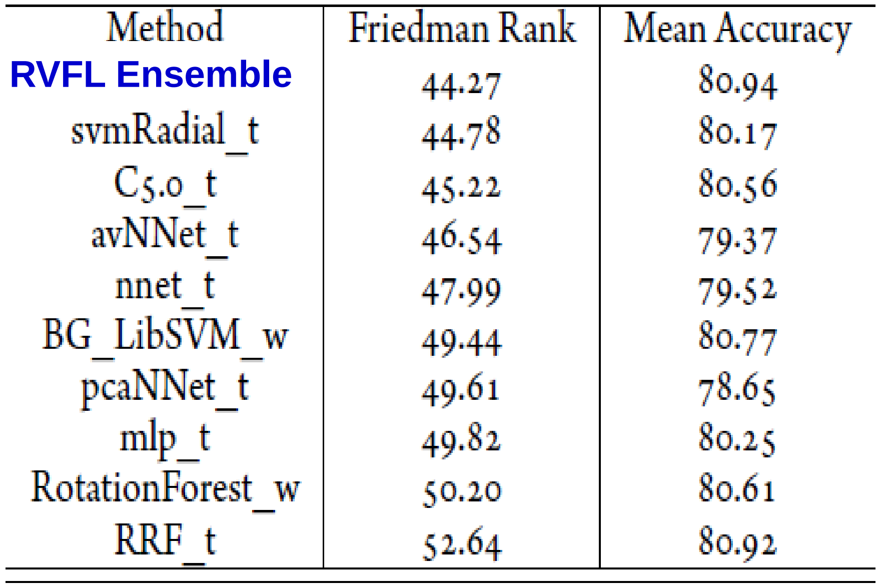
Overall performance compared with JMLR Results[1]

Top 10 classifiers

Method	Friedman Rank	Mean accuracy
MPRaF-P	**33.10**	**82.05**
CoKRR-max	**34.13**	**81.79**
MPRRoF-P	**34.56**	**82.35**
parRF_t	34.97	81.96
rf_t	35.16	82.30
svm_C	36.78	81.77
svmPoly_t	40.36	80.31
KRR / elm_kernel_m	41.65	81.52
Rforest_R	41.67	81.94
svmRadialCost_r	42.47	80.54

Top 11-20 classifier

Sinle RVFL is below #20.



Method	Friedman Rank	Mean Accuracy
RVFL Ensemble	44.27	80.94
svmRadial_t	44.78	80.17
C5.0_t	45.22	80.56
avNNet_t	46.54	79.37
nnet_t	47.99	79.52
BG_LibSVM_w	49.44	80.77
pcaNNet_t	49.61	78.65
mlp_t	49.82	80.25
RotationForest_w	50.20	80.61
RRF_t	52.64	80.92

L. Zhang, P. N. Suganthan, Benchmarking Ensemble Classifiers with Novel Co-trained Kernel Ridge Regression and Random Vector Functional Link Ensembles, IEEE Computational Intelligence Magazine, **Nov. 2017.**

Our proposed methods in blue bold. Other results copied from the JMLR reference:

Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems?." *The Journal of Machine Learning Research* 15.1 (2014): 3133-3181.

NANYANG TECHNOLOGICAL UNIVERSITY

Random Vector Functional Link Neural Network based Ensemble Deep Learning

- We present a deep learning framework based on randomized neural network. In particular, inspired by the principles of Random Vector Functional Link (RVFL) network, we present a deep RVFL network (dRVFL) with stacked layers.
- The parameters of the hidden layers of the dRVFL are randomly generated within a suitable range and kept fixed while the output weights are computed using the closed form solution as in a standard RVFL network.
- We also present an ensemble deep network (edRVFL) that can be regarded as a marriage of ensemble learning with deep learning. Unlike traditional ensembling approaches that require training several models independently from scratch, edRVFL is obtained by training a single dRVFL network once.

Reference:

Random Vector Functional Link Neural Network based Ensemble Deep Learning, R Katuwal, PN Suganthan, M Tanveer, arXiv preprint arXiv:1907.00350

NANYANG TECHNOLOGICAL UNIVERSITY

Deep Random Vector Functional Link Network

- The Deep Random Vector Functional Link (dRVFL) network is an extension of the shallow RVFL network in the context of representation learning or deep learning.
- The dRVFL network is typically characterized by a stacked hierarchy of hidden layers. The input to each layer in the stack is the output of the preceding layer wherein each layer builds an internal representation of the input data.
- Although the stacked hierarchical organization of hidden layers in dRVFL network allows a general flexibility in the size (both in width and depth) of the network, for the sake of simplicity here we consider a stack of *L* hidden layers each of which contains the same number of hidden nodes *N*. For the ease of notation, we omit the bias term in the formulas. The output of the first hidden layer is then defined as follows:

$$\mathbf{H}^{(1)} = g(\mathbf{X}\mathbf{W}^{(1)}) ,$$

while for every layer > 1 it is defined as:

$$\mathbf{H}^{(L)} = g(\mathbf{H}^{(L-1)}\mathbf{W}^{(L)}) ,$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times N}$ and $\mathbf{W}^{(L)} \in \mathbb{R}^{N\times N}$ are the weight matrices between the input-first hidden layer and inter hidden layers respectively. These parameters (weights and biases) of the hidden neurons are randomly generated within a suitable range and kept fixed during the training. $g(\cdot)$ is the non-linear activation function. The input to the output layer is then defined as:

$$\mathbf{D} = [\mathbf{H}^{(1)}\ \mathbf{H}^{(2)} \dots \mathbf{H}^{(L-1)}\ \mathbf{H}^{(L)}\ \mathbf{X}] .$$

The output of the dRVFL network is then defined as follows:

$$Y = \mathrm{D}\beta_d .$$

The output weight $\beta_d \in \mathbb{R}^{(NL+d)\times K}$ (K: the number of classes) is then solved using primal or dual solutions.

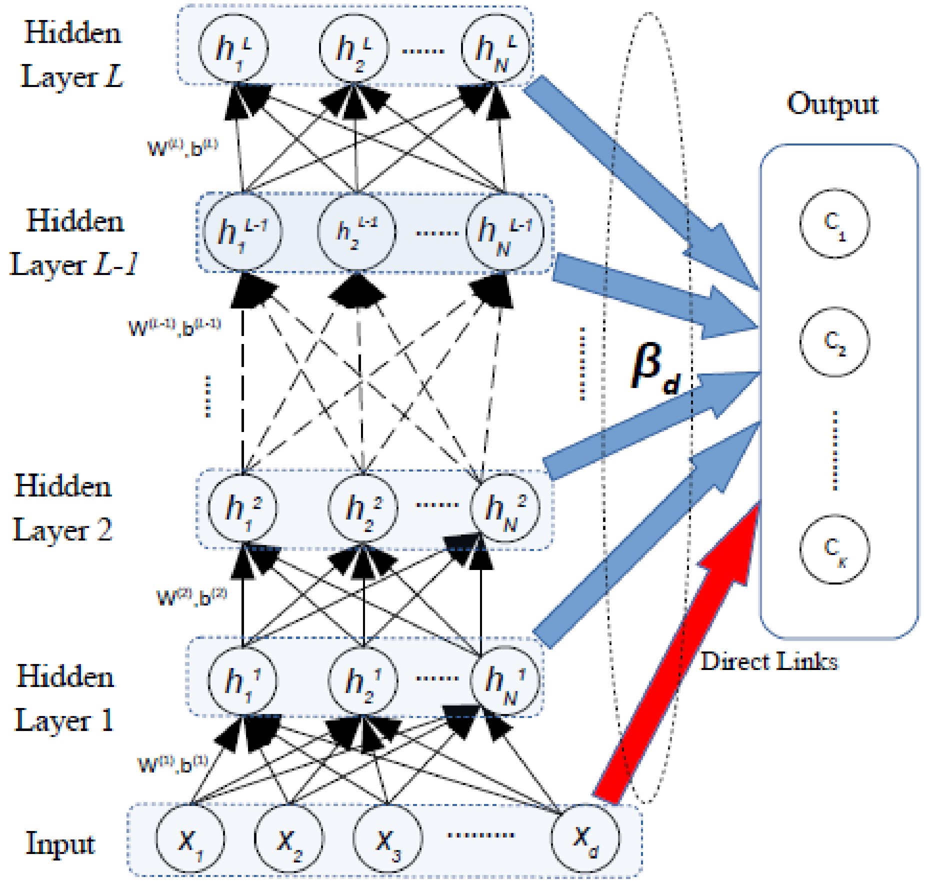


Figure 2: Framework of a dRVFL network. It consists of several hidden layers stacked on top of each other whose parameters (weights and biases between the hidden layers) are randomly generated and kept fixed during the training. Only the output weights β_d need to be computed as in the shallow RVFL network.

Single classifier/classification decision.

Reference:

Random Vector Functional Link Neural Network based Ensemble Deep Learning, R Katuwal, PN Suganthan, M Tanveer, arXiv preprint arXiv:1907.00350

NANYANG TECHNOLOGICAL UNIVERSITY

Ensemble Deep Random Vector Functional Link Network

- The ensemble deep RVFL network serves three purposes: 1) instead of using only the higher level representations (features extracted from the final hidden layer of the data as in a conventional deep learning model for classification, it employs rich intermediate features also for final decision making. 2) the ensemble is obtained by training a single dRVFL network once with a training cost slightly higher than that of a single dRVFL but cheaper than training several independent models of dRVFL. 3) like dRVFL, the edRVFL framework is generic and any RVFL variant can be used with it.

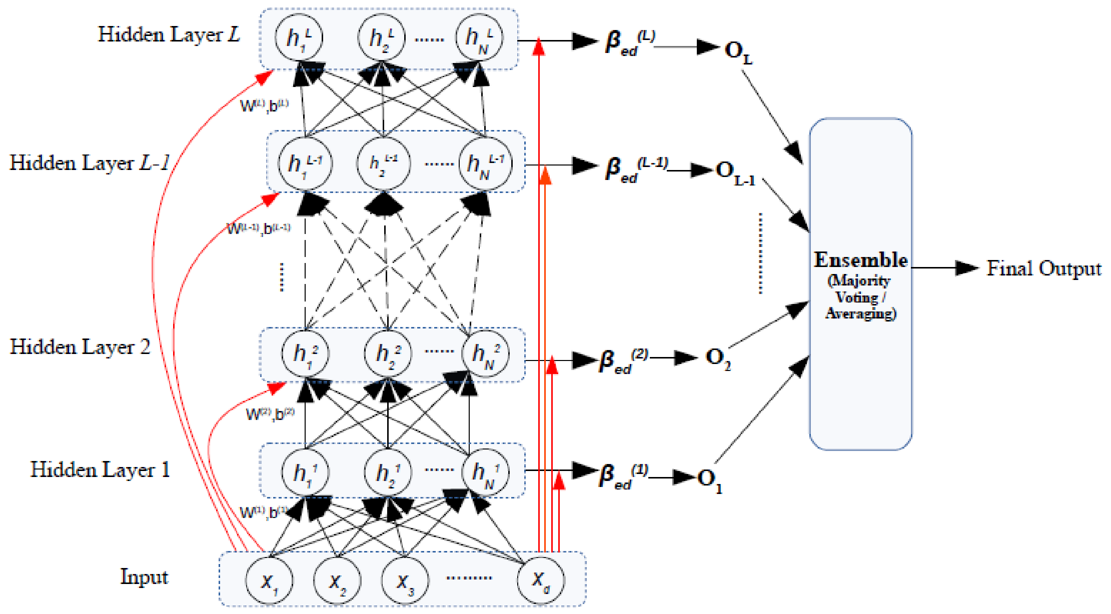


Figure 3: Framework of ensemble deep RVFL network (edRVFL). It differs from dRVFL in that the computation of final output weight β_d is decomposed into several small β_{ed}. Specifically, each small β_{ed} is independently computed (treated as independent model) and the final output is obtained by using either majority voting or averaging of the models. Each higher level model is fed with original input data and the non-linearly transformed features from the preceding model. $\mathbf{O}_1, \ldots, \mathbf{O}_L$ represents the output of each model.

The input to each hidden layer is the non-linearly transformed features from the preceding layer as in dRVFL along with the original input features (direct links) as in standard RVFL. The direct links act as a regularization for the randomization. The input of the first hidden layer is then defined as follows:

$$\mathbf{H}^{(1)} = g(\mathbf{X}\mathbf{W}^{(1)}),$$

while for every layer > 1 it is defined as:

$$\mathbf{H}^{(L)} = g([\mathbf{H}^{(L-1)}\mathbf{X}]\mathbf{W}^{(L)}).$$

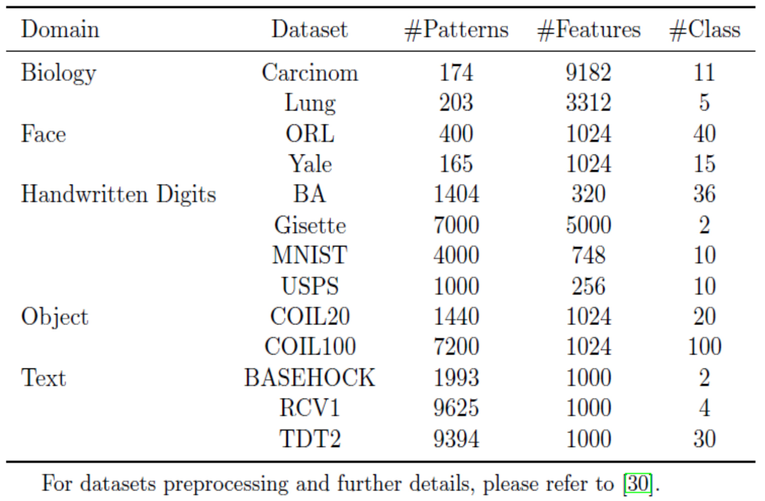
The output weights $\beta_{\boldsymbol{ed}}$ are then solved independently using primal or dual solutions.

Experiments and Results

-O: Without input-output direct links (e.g. ELM)

[30] Y. Zhang, J. Wu, Z. Cai, B. Du, P. S. Yu, An unsupervised parameter learning model for RVFL neural network, Neural Networks 112 (2019) 85-97. doi:10.1016/j.neunet.2019.01.007
[34] J. Tang, C. Deng, G. Huang, Extreme learning machine for multilayer 560 perceptron, IEEE Transactions on Neural Networks and Learning Systems, 27 (4) (2016) 809{821. doi:10.1109/TNNLS.2015.2424995.

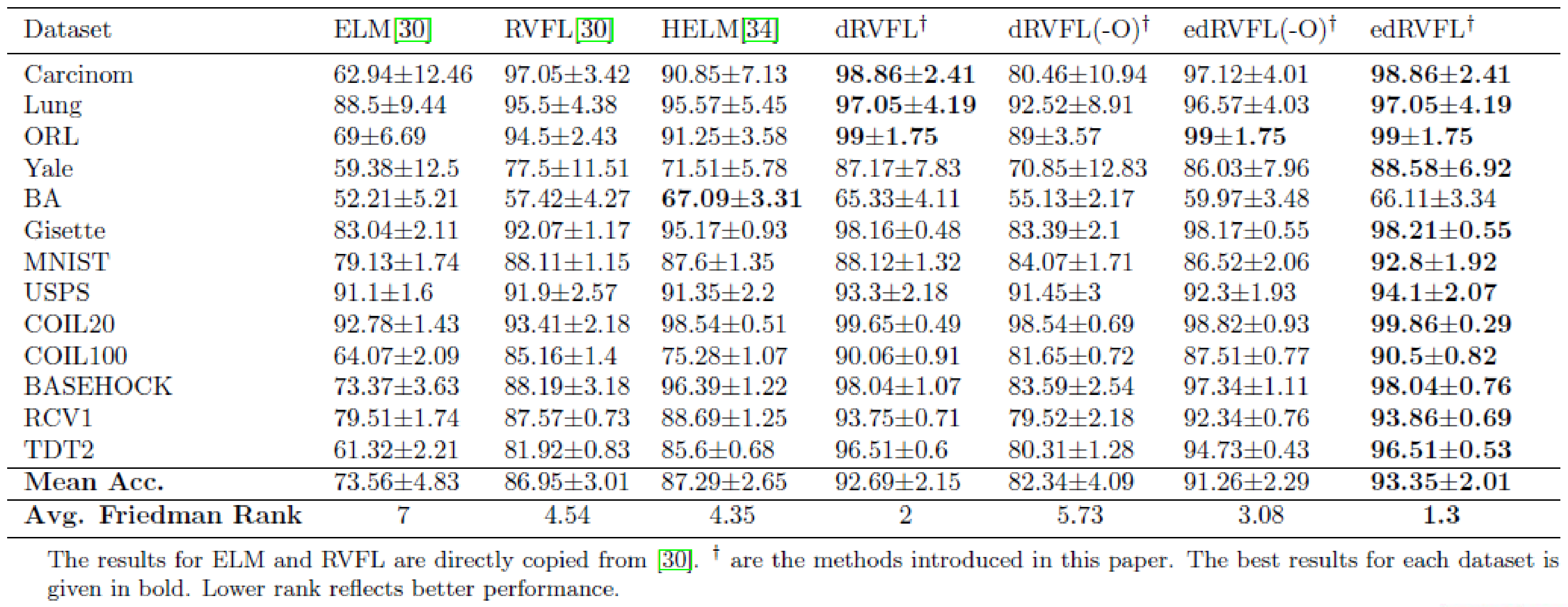
Table 1: Overview of the datasets used in this paper.



Domain	Dataset	#Patterns	#Features	#Class
Biology	Carcinom	174	9182	11
	Lung	203	3312	5
Face	ORL	400	1024	40
	Yale	165	1024	15
Handwritten Digits	BA	1404	320	36
	Gisette	7000	5000	2
	MNIST	4000	748	10
	USPS	1000	256	10
Object	COIL20	1440	1024	20
	COIL100	7200	1024	100
Text	BASEHOCK	1993	1000	2
	RCV1	9625	1000	4
	TDT2	9394	1000	30

For datasets preprocessing and further details, please refer to [30].

Table 2: Accuracy (%) of the standard RVFL based methods on all the datasets

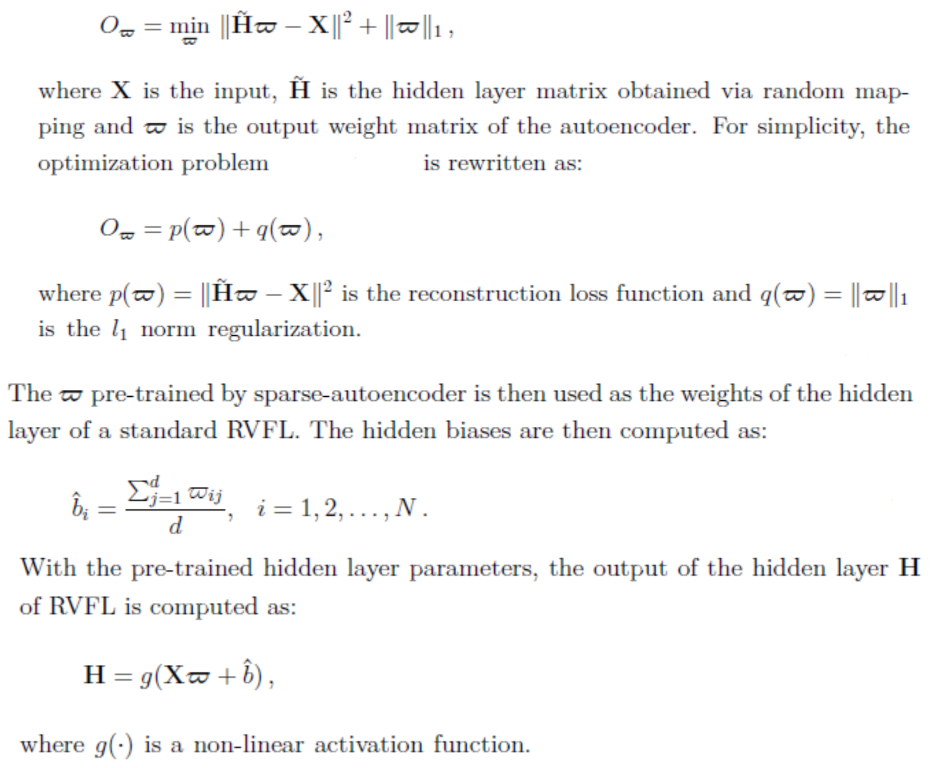


Dataset	ELM[30]	RVFL[30]	HELM[34]	dRVFL[†]	dRVFL(-O)[†]	edRVFL(-O)[†]	edRVFL[†]
Carcinom	62.94±12.46	97.05±3.42	90.85±7.13	**98.86±2.41**	80.46±10.94	97.12±4.01	**98.86±2.41**
Lung	88.5±9.44	95.5±4.38	95.57±5.45	**97.05±4.19**	92.52±8.91	96.57±4.03	**97.05±4.19**
ORL	69±6.69	94.5±2.43	91.25±3.58	**99±1.75**	89±3.57	**99±1.75**	**99±1.75**
Yale	59.38±12.5	77.5±11.51	71.51±5.78	87.17±7.83	70.85±12.83	86.03±7.96	**88.58±6.92**
BA	52.21±5.21	57.42±4.27	**67.09±3.31**	65.33±4.11	55.13±2.17	59.97±3.48	66.11±3.34
Gisette	83.04±2.11	92.07±1.17	95.17±0.93	98.16±0.48	83.39±2.1	98.17±0.55	**98.21±0.55**
MNIST	79.13±1.74	88.11±1.15	87.6±1.35	88.12±1.32	84.07±1.71	86.52±2.06	**92.8±1.92**
USPS	91.1±1.6	91.9±2.57	91.35±2.2	93.3±2.18	91.45±3	92.3±1.93	**94.1±2.07**
COIL20	92.78±1.43	93.41±2.18	98.54±0.51	99.65±0.49	98.54±0.69	98.82±0.93	**99.86±0.29**
COIL100	64.07±2.09	85.16±1.4	75.28±1.07	90.06±0.91	81.65±0.72	87.51±0.77	**90.5±0.82**
BASEHOCK	73.37±3.63	88.19±3.18	96.39±1.22	98.04±1.07	83.59±2.54	97.34±1.11	**98.04±0.76**
RCV1	79.51±1.74	87.57±0.73	88.69±1.25	93.75±0.71	79.52±2.18	92.34±0.76	**93.86±0.69**
TDT2	61.32±2.21	81.92±0.83	85.6±0.68	96.51±0.6	80.31±1.28	94.73±0.43	**96.51±0.53**
Mean Acc.	73.56±4.83	86.95±3.01	87.29±2.65	92.69±2.15	82.34±4.09	91.26±2.29	**93.35±2.01**
Avg. Friedman Rank	7	4.54	4.35	2	5.73	3.08	**1.3**

The results for ELM and RVFL are directly copied from [30]. [†] are the methods introduced in this paper. The best results for each dataset is given in bold. Lower rank reflects better performance.

Sparse pre-trained RVFL (SP-RVFL) [1]

- In a standard RVFL network, the hidden layer parameters are randomly generated within a suitable range and kept fixed thereafter.
- Even though RVFL has demonstrated its efficacy in various domains, its performance is often challenged by randomly assigned hidden layer parameters.
- In an SP-RVFL network, an autoencoder with l_1 regularization is employed to learn the hidden layer parameters. Specifically, the optimization problem for the autoencoder is given by:



$$O_{\varpi} = \min_{\varpi} \|\tilde{\mathbf{H}}\varpi - \mathbf{X}\|^2 + \|\varpi\|_1 \,,$$

where $\mathbf{X}$ is the input, $\tilde{\mathbf{H}}$ is the hidden layer matrix obtained via random mapping and ϖ is the output weight matrix of the autoencoder. For simplicity, the optimization problem is rewritten as:

$$O_{\varpi} = p(\varpi) + q(\varpi) \,,$$

where $p(\varpi) = \|\tilde{\mathbf{H}}\varpi - \mathbf{X}\|^2$ is the reconstruction loss function and $q(\varpi) = \|\varpi\|_1$ is the l_1 norm regularization.

The ϖ pre-trained by sparse-autoencoder is then used as the weights of the hidden layer of a standard RVFL. The hidden biases are then computed as:

$$\hat{b}_i = \frac{\sum_{j=1}^{d} \varpi_{ij}}{d}, \quad i = 1, 2, \ldots, N \,.$$

With the pre-trained hidden layer parameters, the output of the hidden layer $\mathbf{H}$ of RVFL is computed as:

$$\mathbf{H} = g(\mathbf{X}\varpi + \hat{b}) \,,$$

where $g(\cdot)$ is a non-linear activation function.

[1] Y. Zhang, J. Wu, Z. Cai, B. Du, P. S. Yu, An unsupervised parameter learning model for RVFL neural network, Neural Networks 112 (2019) 85-97. doi:10.1016/j.neunet.2019.01.007

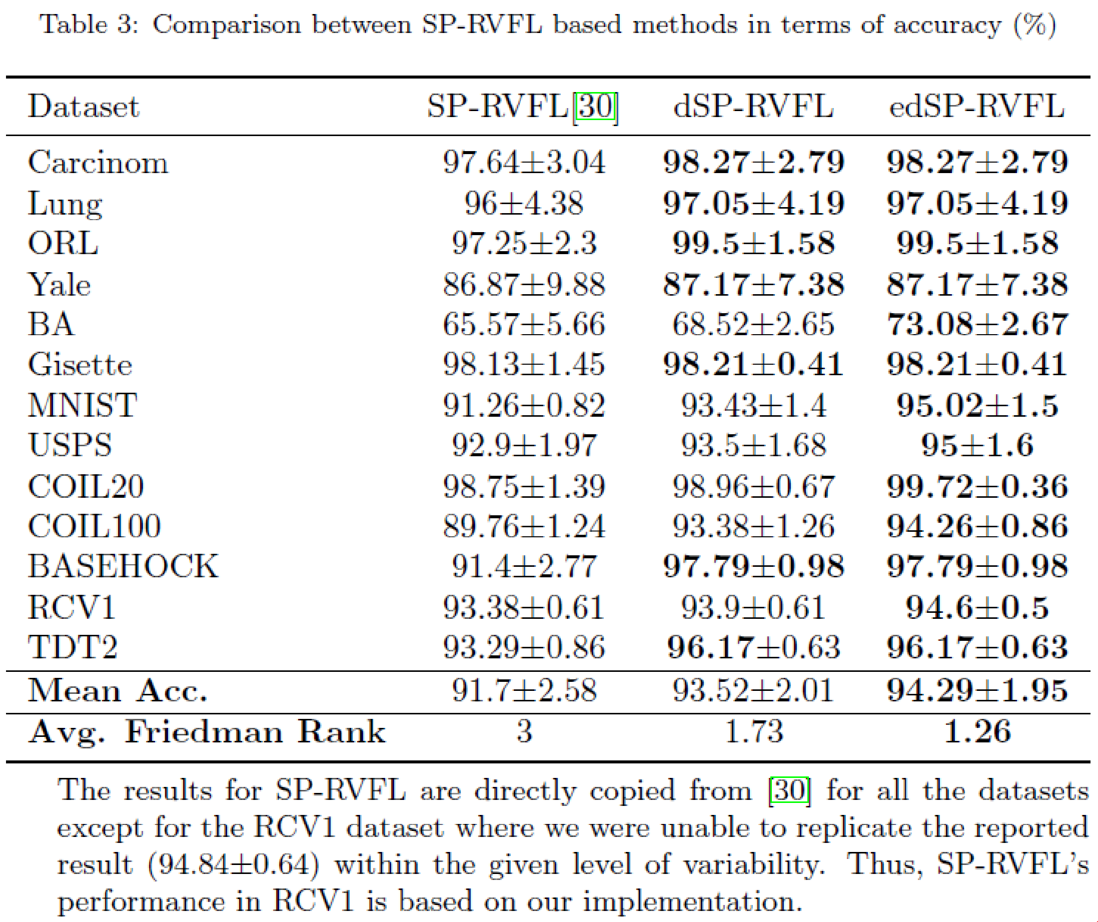


Table 3: Comparison between SP-RVFL based methods in terms of accuracy (%)

Dataset	SP-RVFL[30]	dSP-RVFL	edSP-RVFL
Carcinom	97.64±3.04	**98.27±2.79**	**98.27±2.79**
Lung	96±4.38	**97.05±4.19**	**97.05±4.19**
ORL	97.25±2.3	**99.5±1.58**	**99.5±1.58**
Yale	86.87±9.88	**87.17±7.38**	**87.17±7.38**
BA	65.57±5.66	68.52±2.65	**73.08±2.67**
Gisette	98.13±1.45	**98.21±0.41**	**98.21±0.41**
MNIST	91.26±0.82	93.43±1.4	**95.02±1.5**
USPS	92.9±1.97	93.5±1.68	**95±1.6**
COIL20	98.75±1.39	98.96±0.67	**99.72±0.36**
COIL100	89.76±1.24	93.38±1.26	**94.26±0.86**
BASEHOCK	91.4±2.77	**97.79±0.98**	**97.79±0.98**
RCV1	93.38±0.61	93.9±0.61	**94.6±0.5**
TDT2	93.29±0.86	**96.17±0.63**	**96.17±0.63**
Mean Acc.	91.7±2.58	93.52±2.01	**94.29±1.95**
Avg. Friedman Rank	3	1.73	**1.26**

The results for SP-RVFL are directly copied from [30] for all the datasets except for the RCV1 dataset where we were unable to replicate the reported result (94.84±0.64) within the given level of variability. Thus, SP-RVFL's performance in RCV1 is based on our implementation.

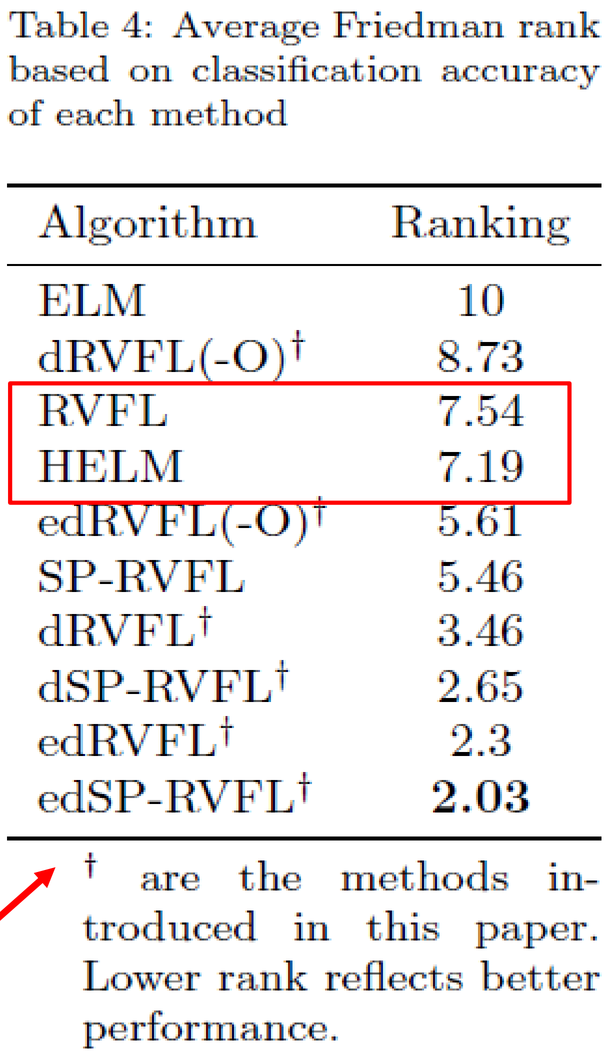


Table 4: Average Friedman rank based on classification accuracy of each method

Algorithm	Ranking
ELM	10
dRVFL(-O)†	8.73
RVFL	7.54
HELM	7.19
edRVFL(-O)†	5.61
SP-RVFL	5.46
dRVFL†	3.46
dSP-RVFL†	2.65
edRVFL†	2.3
edSP-RVFL†	**2.03**

† are the methods introduced in this paper. Lower rank reflects better performance.

-O: Without input-output direct links poor performance.
RVFL (1994) is competitive with HELM (2016) (Red box)

Yoh-Han Pao, et al. "Learning and generalization characteristics of the random vector functional-link net". Neurocomputing **6.2** **(1994), pp. 163–180.**

J. Tang, C. Deng, G. Huang, Extreme learning machine for multilayer 560 perceptron, IEEE Transactions on Neural Networks and Learning Systems, **27 (4) (2016) 809-821**. doi:10.1109/TNNLS.2015.2424995 .

NANYANG TECHNOLOGICAL UNIVERSITY

Summary

- We first presented a deep learning model (dRVFL) based on random vector functional link network. As in a standard RVFL network, the parameters of the hidden layers were randomly generated and kept fixed with the output weights computed analytically using a closed form solution.
- The dRVFL network while extracting rich feature representations through several hidden layers also acts as a weighting network thereby, providing a weight to features from all the hidden layers including the original features obtained via direct links.
- We then proposed an ensemble dRVFL, edRVFL, which combines ensemble learning with deep learning. Instead of training several models independently as in traditional ensembles, edRVFL can be obtained by training a deep network only once. The training cost of edRVFL is slightly greater than that of a single dRVFL network but significantly lower than that of the traditional ensembles.
- Both dRVFL and edRVFL are generic and any RVFL variant can be used with them.
- Extensive experiments on several classification datasets showed that the our deep learning RVFL networks achieve better generalization compared to pertinent randomized neural networks.
- The proposed approach can b integrated with other randomized neural networks such as the SP-RVFL (2019).

Outline

Part I: Classification by

- Neural Networks such as the RVFL
- Kernel Ridge Regression (KRR)
- Random Forest (RF)

Part II: Time Series Forecasting & Classification

(by RVFL, RF, Deep Learners)

Part III: Visual Tracking

(by RVFL, RF, KRR, etc.)

NANYANG TECHNOLOGICAL UNIVERSITY

Linear Ridge Regression

Linear ridge regression :
$$\frac{1}{n}\sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda \| w \|^2$$

Its solution:

$$\sum_i (y_i - \mathbf{w}^T \mathbf{x}_i)\mathbf{x}_i = \lambda \mathbf{w} \quad \Rightarrow \quad \mathbf{w} = \left(\lambda \mathbf{I} + \sum_i \mathbf{x}_i \mathbf{x}_i^T\right)^{-1} \left(\sum_j y_j \mathbf{x}_j\right)$$

Consider the feature mapping below which maps the original feature to higher dimension to enhance the discriminability, i.e. the kernel trick:

$$x_i : \varphi(x_i)$$

Kernel Ridge Regression

According to the representer theorem, one can express the solution as a linear combination of the samples:

$$w = \sum_i a_i \varphi(x_i).$$

Objective with kernel trick:

$$\min_{a} \|Y - Ka\|^2 + \lambda a^T K a;$$

The solution is: $a = (K + \lambda I)^{-1} Y;$

KRR for Classification

- Through the commonly used 0-1 coding:

$$Y_{ij} = \begin{cases} 1 & \text{If } i^{th} \text{ sample belongs to the } j^{th} \text{ class} \\ 0 & \text{Otherwise} \end{cases}$$

KRR for Regression: C. Saunders, A. Gammerman and V. Vovk, "Ridge Regression Learning Algorithm in Dual Variables", in Proc ICML 1998.

KRR for Classification: S. An, W. Liu, and S. Venkatesh, 2007, "Face recognition using kernel ridge regression", in CVPR 2007 : Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, Piscataway, N.J, pp. 1-7, 2007.

KRR name was given to the 1998 ICML work in 2000 or so.

KRR Vs Kernel ELM

- Kernel ELM was derived in 2010 from the ELM proposed in 2004. ELM is a variation of RVFL (1994) without bias and without input-output direct links.
- Codes of Kernel ELM are identical to the codes KRR developed in 1998.

KRR for Regression: C. Saunders, A. Gammerman and V. Vovk, "Ridge Regression Learning Algorithm in Dual Variables", in Proc ICML 1998.

KRR for Classification: S. An, W. Liu, and S. Venkatesh, 2007, "Face recognition using kernel ridge regression", in CVPR 2007 : Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, Piscataway, N.J, pp. 1-7, 2007.

Kernel ELM: Optimization based extreme learning machine for classification, Neurocomputing 74 (2010) 155–163.

Kernel Trick for Neural Networks?

- Kernel trick is commonly applied to linear methods to have nonlinear characteristics. Examples are kernel PCA, Kernel LDA, Kernel Ridge Regression (KRR), etc.

- Have we seen Kernel trick for neural nets commonly? Neural Nets usually have nonlinear activations. Opposite seems meaningful, i.e. kernel methods mimicking neural networks …

- Is it meaningful to apply the kernel trick to ELM as follows?

Output = $dW = dD^T(\lambda I + DD^T)^{-1}Y$ (total features > # training samples)

Kernel Trick for ELM: Just replace DD^T by kernel matrix based on training data while dD^T is the kernel mapping between the testing sample and training samples.

Questions to ask after applying the kernel trick:

- Now do we still have any randomization in the above solution? No
- Can we see the neural network structure such as the hidden neurons? No
- In this case, do we need to talk about fixed randomized input weights, closed form solution for output weights, etc. in the context of ELM ? No
- Is there a straightforward way to obtain the above Kernel Realization? **Yes, via KRR, instead of via ELM !!**

Co-trained KRR / Positive Correlation Learning

- One KRR sometimes not enough: features not good, kernel not good, parameters not good.
- Multi-Kernel Learning: Optimizing the combination parameter of kernels (high computational complexity, usually iterative)

Reference: L. Zhang, P. N. Suganthan, Benchmarking Ensemble Classifiers with Novel Co-trained Kernel Ridge Regression and Random Vector Functional Link Ensembles, IEEE Computational Intelligence Magazine, Nov 2017.

Co-trained KRR / Positive Correlation Learning

- Negative Correlation (i.e. negative correlated evolutionary ensemble neural networks) is better for an ensemble of unstable classifiers while positive correlation (or consensus) is better for an ensemble of stable classifiers such as the KRR.
- Consensus: Make 2 different kernels classifiers to have the same results, i.e. Positive Correlation Based Learning.

$$E(\alpha) = \min_{\alpha} \sum_{i=1}^{2} (||\mathbf{Y} - \mathbf{K}_i \alpha_i||^2 + C_1 \alpha_i^T \mathbf{K}_i \alpha_i)$$

$$+ C_2 \sum_{i,j=1}^{2} ||\mathbf{K}_i \alpha_i - \mathbf{K}_j \alpha_j||^2 .$$

Co-trained KRR / Positive Correlation Learning

- Extensive testing showed no improvement in classification accuracy on UCI datasets with more than 2 kernels.
- Computational cost, parameter tuning increase rapidly with further kernels.
- In addition to using different kernel parameters, original training features were rotated too.
- Two kernels:

$$\mathbf{G}_i = (2C_2 + 1)\mathbf{K}_i\mathbf{K}_i + C_1\mathbf{K}_i,$$

$$\begin{aligned}\alpha_1 =& (\mathbf{G}_1 - 4C_2^2\mathbf{K}_1\mathbf{K}_2\mathbf{G}_2^{-1}\mathbf{K}_2\mathbf{K}_1)^{-1} \\ & \times (\mathbf{K}_1\mathbf{Y} + 2C_2\mathbf{K}_1\mathbf{K}_2\mathbf{G}_2^{-1}\mathbf{K}_2\mathbf{Y}), \\ \alpha_2 =& (\mathbf{G}_2 - 4C_2^2\mathbf{K}_2\mathbf{K}_1\mathbf{G}_1^{-1}\mathbf{K}_1\mathbf{K}_2)^{-1} \\ & \times (\mathbf{K}_2\mathbf{Y} + 2C_2\mathbf{K}_2\mathbf{K}_1\mathbf{G}_1^{-1}\mathbf{K}_1\mathbf{Y}).\end{aligned}$$

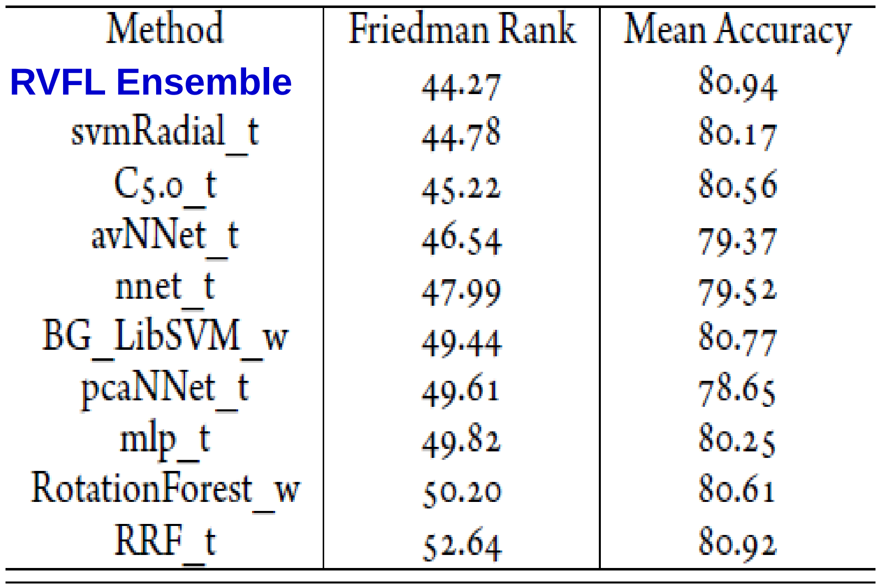
NANYANG TECHNOLOGICAL UNIVERSITY

Overall performance compared with JMLR Results[1]

Top 10 classifiers

Method	Friedman Rank	Mean accuracy
MPRaF-P	33.10	82.05
CoKRR-max	34.13	81.79
MPRRoF-P	34.56	82.35
parRF_t	34.97	81.96
rf_t	35.16	82.30
svm_C	36.78	81.77
svmPoly_t	40.36	80.31
KRR / elm_kernel_m	41.65	81.52
Rforest_R	41.67	81.94
svmRadialCost_r	42.47	80.54

Top 11-20 classifier



Method	Friedman Rank	Mean Accuracy
RVFL Ensemble	44.27	80.94
svmRadial_t	44.78	80.17
C5.0_t	45.22	80.56
avNNet_t	46.54	79.37
nnet_t	47.99	79.52
BG_LibSVM_w	49.44	80.77
pcaNNet_t	49.61	78.65
mlp_t	49.82	80.25
RotationForest_w	50.20	80.61
RRF_t	52.64	80.92

L. Zhang, P. N. Suganthan, Benchmarking Ensemble Classifiers with Novel Co-trained Kernel Ridge Regression and Random Vector Functional Link Ensembles, IEEE Computational Intelligence Magazine, Nov 2017.

Our proposed methods in blue bold. Other results copied from the JMLR reference:

Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems?." *The Journal of Machine Learning Research* 15.1 (2014): 3133-3181.

2nd best performance by CoKRR

NANYANG TECHNOLOGICAL UNIVERSITY

Kernel Ridge Regression (KRR)

- KRR $\longrightarrow$ inversion of kernel matrix $\rightarrow$ $O(N^3)$ time and $O(N^2)$ memory where N is the number of training samples.
- out-of-memory issues
- Solution: Kernel or Gram matrix approximation
  1. Nystrom Method[1]
  2. Random Fourier Features[2]
  3. Fast-KRR[3]
  4. …….

[1] C. Williams and M.Seeger, Using the nystrom method to speed up kernel machines, *NIPS*, pages 682-688,2001
[2] A. Rahimi and B. Recht, Random features for large-scale kernel machines, *NIPS*, pages 1177-1184, 2007
[3] Y. Zhang, J. Duchi, and M. Wainwright, Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates, *JMLR*, 2015

NANYANG TECHNOLOGICAL UNIVERSITY

Speed-up for Kernel Ridge Regression

Nystrom Method

- Approximates the kernel matrix by a low rank matrix.
- Randomly samples a subset of training examples and computes a kernel matrix $\widehat{K}$ for the random samples.
- Then represents each data point by a vector based on its kernel similarity to the random samples and the sampled kernel matrix $\widehat{K}$.

The low-rank matrix $\hat{K}$ is obtained by: $\hat{K} = K_{N,n}K_{n,n}^{-1}K_{n,N}$ where n is the rows/columns of K chosen randomly without replacement, $K_{N,n}$ is the $N \times n$ block of the original matrix K.

- Computation complexity reduced to $O(n^2N)$.

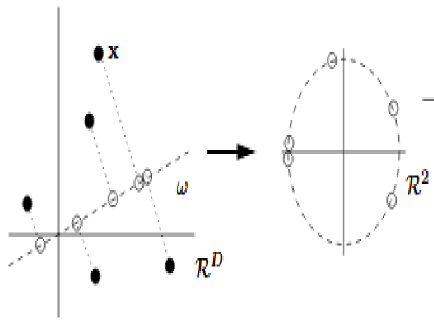
Commonly *n* is square root of *N*.

NANYANG TECHNOLOGICAL UNIVERSITY

Speed-up for Kernel Ridge Regression

Random Fourier Features

- A shift-invariant kernel is the Fourier transform of a non-negative measure.
- Using the above property, represent each data point by a low-dimensional random Fourier features and then apply existing fast linear methods. Avoid implicit lifting provided by the kernel trick.



Kernel Name	$k(\Delta)$	$p(\omega)$
Gaussian	$e^{-\frac{\|\Delta\|_2^2}{2}}$	$(2\pi)^{-\frac{D}{2}} e^{-\frac{\|\omega\|_2^2}{2}}$
Laplacian	$e^{-\|\Delta\|_1}$	$\prod_d \frac{1}{\pi(1+\omega_d^2)}$
Cauchy	$\prod_d \frac{2}{1+\Delta_d^2}$	$e^{-\|\Delta\|_1}$

Figure 1: Random Fourier Features. Each component of the feature map $\mathbf{z}(\mathbf{x})$ projects $\mathbf{x}$ onto a random direction ω drawn from the Fourier transform $p(\omega)$ of $k(\Delta)$, and wraps this line onto the unit circle in $\mathcal{R}^2$. After transforming two points $\mathbf{x}$ and $\mathbf{y}$ in this way, their inner product is an unbiased estimator of $k(\mathbf{x},\mathbf{y})$. The mapping $z(\mathbf{x}) = \cos(\omega'\mathbf{x} + b)$ additionally rotates this circle by a random amount b and projects the points onto the interval $[0,1]$. The table lists some popular shift-invariant kernels and their Fourier transforms. To deal with non-isotropic kernels, we can first whiten the data and apply one of these kernels

```
Algorithm 1 Random Fourier Features.
Require: A positive definite shift-invariant kernel $k(\mathbf{x},\mathbf{y}) = k(\mathbf{x}-\mathbf{y})$.
Ensure: A randomized feature map $\mathbf{z}(\mathbf{x}) : \mathcal{R}^d \to \mathcal{R}^D$ so that $\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y}) \approx k(\mathbf{x}-\mathbf{y})$.
  Compute the Fourier transform $p$ of the kernel $k$: $p(\omega) = \frac{1}{2\pi}\int e^{-j\omega'\delta}k(\delta)\, d\Delta$.
  Draw $D$ iid samples $\omega_1,\cdots,\omega_D \in \mathcal{R}^d$ from $p$ and $D$ iid samples $b_1,\ldots,b_D \in \mathcal{R}$ from the uniform distribution on $[0,2\pi]$.
  Let $\mathbf{z}(\mathbf{x}) \equiv \sqrt{\frac{2}{D}}\left[\cos(\omega_1'\mathbf{x}+b_1) \cdots \cos(\omega_D'\mathbf{x}+b_D)\right]'$.
```

NANYANG TECHNOLOGICAL UNIVERSITY

Nystrom and Random Fourier Features

- Woodbury approximation is used alongside low-rank approximations to efficiently (and approximately) invert kernel matrices.
- The Woodbury approximation is given by:

$$(\mathbf{A}+\mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1}+\mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1},$$

where $\mathbf{A} = \lambda\mathbf{I}$ and $\widetilde{\mathbf{K}} = \mathbf{BCD}$ in the context of the Nyström method.

- B = $K_{N,n}$, C = $K^{-1}{}_{n,n}$ and D = $K_{n,N}$ from $\hat{K} = K_{N,n}K_{n,n}^{-1}K_{n,N}$
- The matrix inversion is reduced from N x N to n x n.

Dimensionality is N x N , total no of training samples

- <u>For Random Fourier Features</u>,
  – $(K+\lambda I)^{-1}$ becomes $(z^T z+\lambda I)^{-1}$ where z is of size N x D. The matrix inversion is reduced to D x D.

Speed-up for Kernel Ridge Regression

Fast-KRR

- randomly partition a data set of size *N* into *m* subsets of equal size, compute an independent kernel ridge regression estimator for each subset, then average (majority vote) the local solutions.
- Proposed for regression problems.

1. Divide the set of samples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ evenly and uniformly at randomly into the m disjoint subsets $S_1, \ldots, S_m \subset \mathcal{X} \times \mathbb{R}$.
2. For each $i = 1, 2, \ldots, m$, compute the *local KRR estimate*
$$\widehat{f}_i := \underset{f \in \mathcal{H}}{\operatorname{argmin}} \left\{ \frac{1}{|S_i|} \sum_{(x,y) \in S_i} (f(x) - y)^2 + \lambda \|f\|_{\mathcal{H}}^2 \right\}.$$
3. Average together the local estimates and output $\bar{f} = \frac{1}{m} \sum_{i=1}^{m} \widehat{f}_i$.

Nystrom Method vs Random Fourier Features vs Fast KRR

- Million Song Dataset [1]:
  - 463,715 training examples, 51,630 testing examples. The task is to predict the year in which the song was released.
- Results taken from [2].
  - Possible to improve the performance of Nystrom and Random Fourier Features based method by advanced sampling techniques other than random sampling [3].

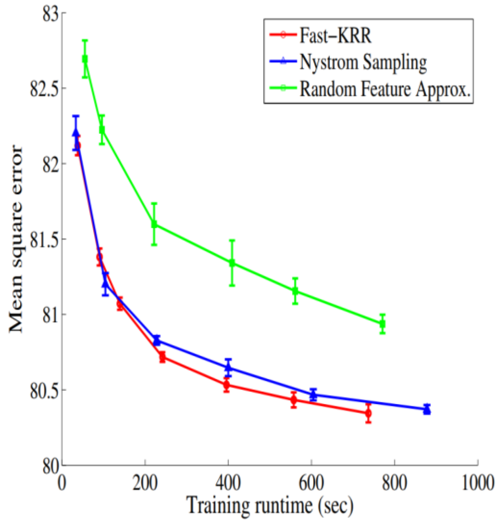


Figure : Results on year prediction on held-out test songs for Fast-KRR, Nyström sampling, and random feature approximation. Error bars indicate standard deviations over ten experiments.

[1] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. *In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR),* 2011
[2] Y. Zhang, J. Duchi, and M. Wainwright, Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates, *JMLR*, 2015
[3] Kumar, S., Mohri, M., & Talwalkar, A. Sampling methods for the Nyström method. *Journal of Machine Learning Research*, *13*(Apr) 2012, 981-1006.

NANYANG TECHNOLOGICAL UNIVERSITY

Kernel Methods for Deep Learning[1]

- How can kernel methods replicate deep learning capabilities?
- A new kernel function proposed to mimic the computation in large, multilayer neural nets is the *n*th order arc-cosine kernel function:

$$k_n(\mathbf{x},\mathbf{y}) = 2\int d\mathbf{w}\, \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\, \Theta(\mathbf{w}\cdot\mathbf{x})\, \Theta(\mathbf{w}\cdot\mathbf{y})\, (\mathbf{w}\cdot\mathbf{x})^n\, (\mathbf{w}\cdot\mathbf{y})^n$$

$$k_n(\mathbf{x},\mathbf{y}) = \frac{1}{\pi}\|\mathbf{x}\|^n\|\mathbf{y}\|^n J_n(\theta) \qquad J_n(\theta) = (-1)^n(\sin\theta)^{2n+1}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi-\theta}{\sin\theta}\right)$$

$$\begin{aligned} J_0(\theta) &= \pi-\theta \\ J_1(\theta) &= \sin\theta + (\pi-\theta)\cos\theta \\ J_2(\theta) &= 3\sin\theta\cos\theta + (\pi-\theta)(1+2\cos^2\theta) \end{aligned}$$

[1] Cho, Y., & Sual, L.K. Kernel Methods for Deep Learning, NIPS 2009

Kernel Methods for Deep Learning

Computation in multilayer threshold networks:

- If the base kernel function *k(x, y)* mimics the computation in a single- layer network, then the iterated mapping should mimic the computation in a multi-layer network.
- The new kernel function which computes the inner product after *l* successive applications of the nonlinear mapping Φ(.) can be written as:

$$k^{(\ell)}(\mathbf{x},\mathbf{y}) = \underbrace{\Phi(\Phi(\ldots\Phi(\mathbf{x})))}_{\ell \text{ times}} \cdot \underbrace{\Phi(\Phi(\ldots\Phi(\mathbf{y})))}_{\ell \text{ times}}$$

- New kernel is required because such iterated mapping for commonly used kernels (linear, polynomial, gaussian) doesn't represent interesting computation and thus, no gain in performance.

For example, consider two-fold composition that maps x to $\Phi(\Phi(x))$. For linear kernels $k(x, y) = x \cdot y$, the composition is trivial: we obtain the identity map $\Phi(\Phi(x)) = \Phi(x) = x$.

- The *l*-fold composition for arc-cosine kernel functions is given in inductive step as:

$$k_n^{(l+1)}(\mathbf{x},\mathbf{y}) = \frac{1}{\pi}\left[k_n^{(l)}(\mathbf{x},\mathbf{x})\,k_n^{(l)}(\mathbf{y},\mathbf{y})\right]^{n/2} J_n\!\left(\theta_n^{(\ell)}\right),$$

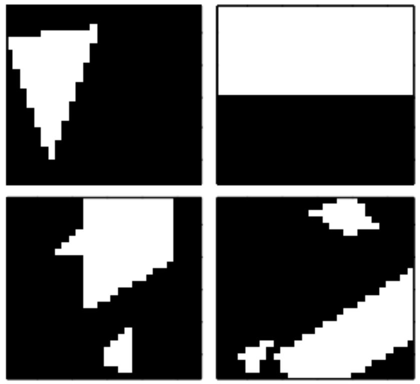
where $\theta_n^{(\ell)}$ is the angle between the images of **x** and **y** in the feature space induced by the ℓ-fold composition. In particular, we can write:

$$\theta_n^{(\ell)} = \cos^{-1}\left(k_n^{(\ell)}(\mathbf{x},\mathbf{y})\left[k_n^{(\ell)}(\mathbf{x},\mathbf{x})\,k_n^{(\ell)}(\mathbf{y},\mathbf{y})\right]^{-1/2}\right).$$

- The resulting kernels mimic the computations in large multi-layer threshold networks.

Kernel Methods for Deep Learning

- *convex* dataset: each dataset contains a white region, and the task is to determine whether the white region is convex (binary classification).
- Results copied from [1].



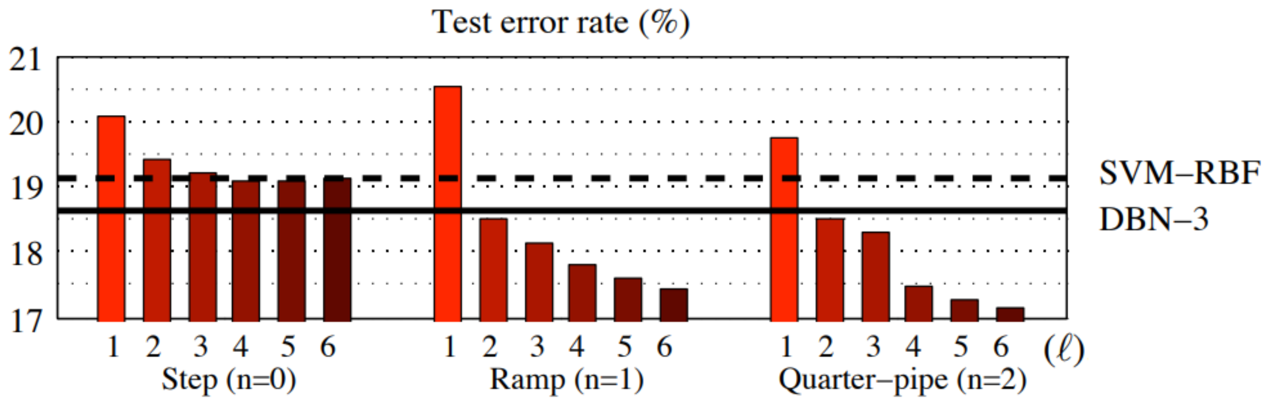


Figure 3: *Left*: examples from the *convex* data set. *Right*: classification error rates on the test set. SVMs with arc-cosine kernels have error rates from 17.15–20.51%. Results are shown for kernels of varying degree (n) and levels of recursion (ℓ). The best previous results are 19.13% for SVMs with RBF kernels and 18.63% for deep belief nets [10]. See text for details.

[1] Cho, Y., & Sual, L.K. Kernel Methods for Deep Learning, *NIPS* 2009 58

NANYANG TECHNOLOGICAL UNIVERSITY

Thank You…

Questions?