Introduction

- Computational Learning Theory
- Empirical Risk minimization
- Loss functions and Structural Risk Minimization
- Topics Covered

Computational Learning Theory (CLT)

- CLT deals with theoretical questions such as:
  - Under what conditions successful learning possible / impossible.
  - Under what conditions a particular learning algorithm is assured of learning successfully, etc.
- Possible approaches are:
  - PAC (Probably Approximately Correct) Learning Framework: Identify classes of hypotheses that can / cannot be learned from a polynomial number of training examples.
  - Mistake Bound Framework: How many training examples will the learner misclassify before converging to a successful hypothesis?

Important Theoretical Questions

- Is it possible to identify classes of learning problems that are inherently difficult or easy, independent of the learning algorithm?
- Can one characterize the number of training examples necessary or sufficient to assure successful learning?
- Can one characterize the number of mistakes that a learner will make before learning the target function?
- Can one characterize the inherent computational complexity of a class of learning algorithms?
- CLT researchers attempts to answer such theoretical issues …

Empirical Risk minimization (ERM)

- The risk of using a machine learning algorithm cannot be in general precisely given as practical problems are defined using data examples/samples, not by precise distributions.
- Without the knowledge of precise distributions, true risk of using an algorithm cannot be computed.
- But, empirical risk can be obtained using sample data.
- Hence, our learning objective is to minimise the empirical risk on the training data.
- Here risk represents wrong decisions / results yielded by the machine learning algorithm.

Loss Functions and Generalization

- There are numerous loss functions for different scenarios such as two-class/multi-class classification, function approximation, time series forecasting, and so on.
- The structure or capacity or complexity of the machine learning method should be appropriate for the complexity of the problem and availability of training data.
- If the structure or capacity or complexity of the machine learning method is too large / small the complexity of the problem and availability of training data, the resulting trained model will be an over-fit / under-fit. In these situations, the model will not perform satisfactorily on test data.
- When the structure or capacity or complexity of the machine learning method should be appropriate, we can expect a good performance on test data. This is also known as good generalisation.

Some Loss Functions

Output yielded by an ML model for an input example x_i is denoted as $h_w(x_i)$, where w represents parameters of the machine learning model.

- Two class classification with true class label y_i taking either +1 or -1:
  - Actual Classification error counts: $\delta(sign(h_w(x_i)) \neq y_i)$. Take note that this delta function takes either 1 (for wrong classification) or 0 (for correct classification)
  - Exponential loss: $e^{-h_w(x_i)y_i}$. If the output of the ML model and true class label have opposite signs, the loss will exponentially increase.
- Loss for function Approximation (Outcomes/desired values are floating numbers):
  - Squared Loss: $(h_w(x_i) - y_i)^2$
  - Absolute Loss: $|h_w(x_i) - y_i|$

Model Tuning and Regularisation

- Complexity of the model has to be tuned to yield good performance. For example, the number of neurons in the hidden layers of a deep neural networks, the number of layers in a deep neural networks, etc.

- In addition to tuning the complexity of the model, parameters of the model has to be regularised too:
  - $L_2 - Regularization$: $r(w) = w^T w = (||w||_2)^2$
  - $L_1 - Regularization$: $r(w) = ||w||_1$

- We will make use of these observations in some of the machine learning models.

Topics Covered

Included in Quiz on 13th April

- Knn Classifier
- Randomization Based Shallow and Deep Neural Networks
- Kernel Ridge Regression
- Random Forests
- Unsupervised Learning

Not Included in Quiz on 13th April

- Linear Classifiers
- Oblique Random Forests

k-Nearest Neighbour

- We are concerned with the following problem: we wish to label some observed pattern x with some class category θ.
- Two possible situations with respect to x and θ may occur:
  - We may have complete statistical knowledge of the distribution of observation x and category θ. In this case, a standard Bayes analysis yields an optimal decision procedure.
  - We may have no knowledge of the distribution of observation x and category θ aside from that provided by pre-classified samples. In this case, a decision to classify x into category θ will depend only on a collection of correctly classified samples.
- The nearest neighbor rule is concerned with the latter case. Such problems are classified in the domain of non-parametric statistics. No optimal classification procedure exists with respect to all underlying statistics under such conditions.

- Here, θ plays the same role as y_i

The NN Rule

- Recall that the only knowledge we have is a set of *i* points of *x* correctly classified into categories θ: (x_i,θ_i). By intuition, it is reasonable to assume that observations that are close together -- for some appropriate metric -- will have the same classification.
- Thus, when classifying an unknown sample x, it seems appropriate to weight the evidence of the nearby x_i's heavily.
- One simple non-parametric decision procedure of this form is the nearest neighbor rule or NN-rule. This rule classifies x in the category of its nearest neighbor. More precisely, we call

$x_m \in (x_1, x_2, x_3, \ldots, x_n)$

x_m is a nearest neighbor to x if

$\min d(x_i, x) = d(x_m, x)$ where $i = 1, 2, \ldots, n$.

The nearest neighbor rule chooses to classify x to the category θ_m, where x_m is the nearest neighbor to x and belongs to class θ_m. A mistake is made if θ_m is not the same as θ which is the true class that x belongs to. Also, notice that the NN-rule only uses the nearest neighbor x_m as a classifier, while ignoring the remaining pre-labeled data points.

Figure 1: The NN rule

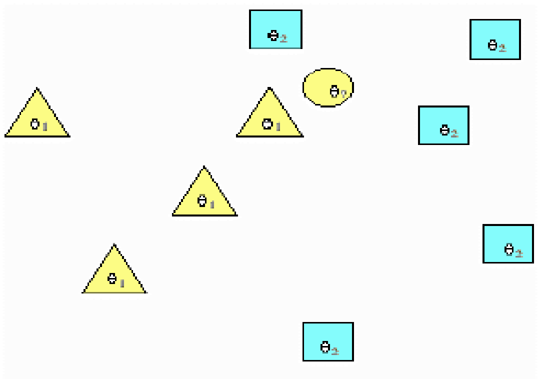


Figure 1 shows an example of the NN rule. In this problem, there are two classes: (yellow triangles) and (blue squares). The circle represents the unknown sample x and as its nearest neighbor comes from class x_1^1, it is labeled as class θ_1.

The k-NN Rule

If the number of pre-classified points is large it makes good sense to use, instead of the single nearest neighbor, the majority vote of the nearest k neighbors. This method is referred to as the k-NN rule.

The number k should be:

1) large to minimize the probability of misclassifying x.

2) small (with respect to the number of samples) so that the points are close enough to x to give an accurate estimate of the true class of x.

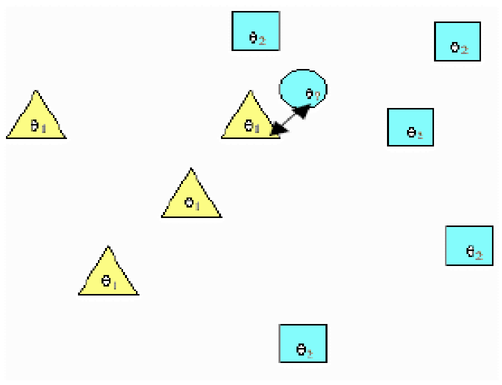


Figure 2: The k-NN rule with k=3

Figure 2 shows an example of the k-NN rule, with k=3. As before, there are two classes: θ_1 (yellow triangles) and θ_2 (blue squares). The circle represents the unknown sample x and as two of its nearest neighbors come from class θ_2, it is labeled class θ_2.

Random Vector Functional Link (RVFL) Neural Networks

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

https://www3.ntu.edu.sg/home/epnsugan/

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

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 for single hidden layer neural net:

  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

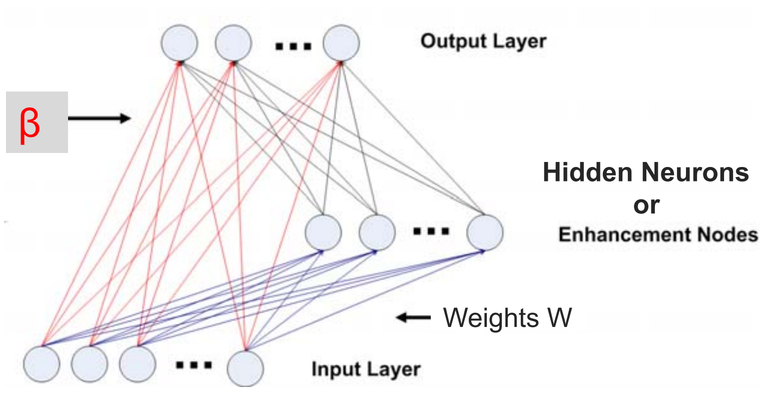
The next slide lists the independent publications on Randomization Based Neural Nets, i.e. there is no citation of each other in all of them in the listing.

- **[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. [& also in "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. 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.
- Widrow B, Greenblatt A, Kim Y, Park D, "The No-Prop algorithm: a new learning algorithm for multilayer neural networks," **Neural Networks. 2013 pp.182-8**.

RVFL and ELM became popular with many follow-up publications.

Structure of RVFL (SLFN with direct links)

Weights Connecting → To output



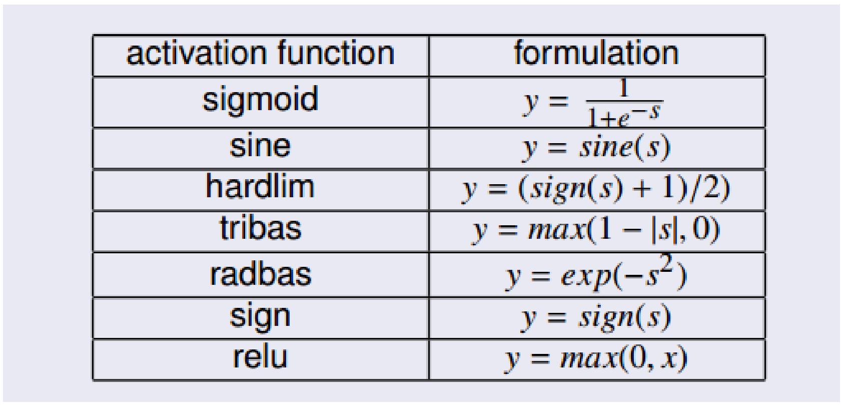
Parameters w (indicated in blue) in enhancement (or hidden layer) nodes are randomly generated in a proper range and kept fixed.

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

Learning aims at ideally obtaining yi = di ∗ β , i = 1, 2, ..., n, where n is the number of training sample, β (indicated in red and grey) are the weights in the output layer, and y, d represent the output and the combined features, respectively.

RVFL details

- Notations:
- X = [x1, x2, ...xn]' : input data (n samples and m features).
- W = [W1,W2, ...Wm]' : 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 ∗ W + 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	$y = max(1 - \lvert s \rvert, 0)$
radbas	$y = exp(-s^2)$
sign	$y = sign(s)$
relu	$y = max(0, x)$

RVFL details

- $$H = \begin{bmatrix} h(x_1) \\ h(x_2) \\ \vdots \\ h(x_n) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_k(x_1) \\ h_1(x_2) & \cdots & h_k(x_2) \\ \vdots & \ddots & \vdots \\ h_1(x_n) & \cdots & h_k(x_n) \end{bmatrix}$$
- $$D = [H, X] = \begin{bmatrix} h_1(x_1) & \cdots & h_k(x_1) & x_{11} & \cdots & x_{1m} \\ h_1(x_2) & \cdots & h_k(x_2) & x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ h_1(x_n) & \cdots & h_k(x_n) & x_{n1} & \cdots & x_{nm} \end{bmatrix} = \begin{bmatrix} d(x_1) \\ d(x_2) \\ \cdots \\ \cdots \\ \cdots \\ d(x_n) \end{bmatrix}$$
- 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 W and b are generated in proper range, the learning of RVFL aims at solving the following problem: $o_i = d_i * \beta$, i = 1, 2, ..., n, where n is the number of training sample, β are the weights in the output layer, and o, d represent the output and the combined features.

- Objective: $\min_{W} ||D\,\beta - Y||^2 + \lambda\, ||\,\beta\,||^2$

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

(**Sometimes 1/C is used instead of λ**)

where λ is the regularization parameter, when $\lambda \to 0$, 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]:

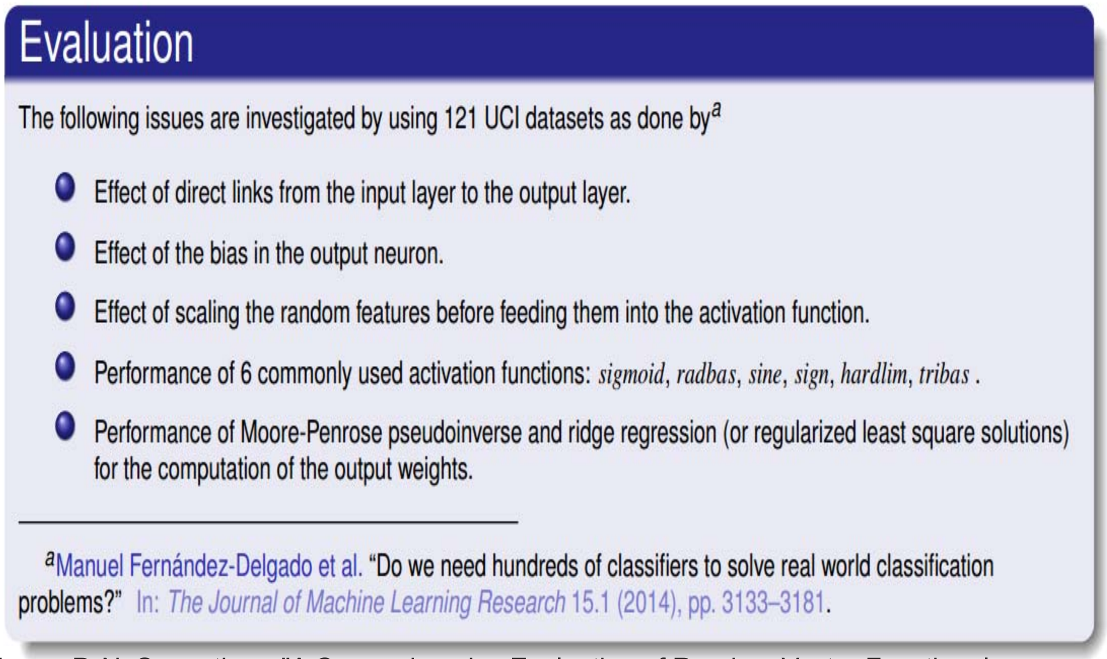
- Formulates a limit-integral representation of the function to be approximated
- Evaluates the integral with the Monte-Carlo Method

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.

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

EVALUATION PROTOCOL

Evaluation Metric: Friedman Ranking [6]

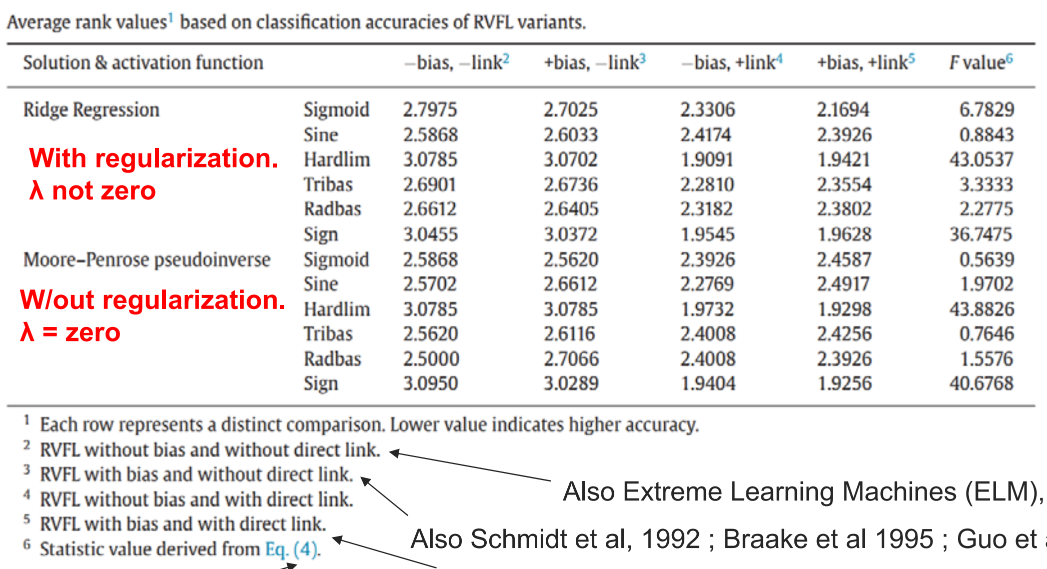
- Rank the algorithms for each dataset separately (the best performing algorithm getting the rank of 1, the second best rank 2 and so on). In case of ties, average ranks are assigned.
- Let *rji* be the rank of the *jth* of *k* algorithms on the *ith* of *N* datasets. The Friedman test compares the average 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 √2. α is the significance level and is set to be 0.05 in this work

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
With regularization. λ not zero	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
W/out regularization. λ = zero	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

[1] Each row represents a distinct comparison. Lower value indicates higher accuracy.
[2] RVFL without bias and without direct link. — Also Extreme Learning Machines (ELM), 2004
[3] RVFL with bias and without direct link. — Also Schmidt et al, 1992 ; Braake et al 1995 ; Guo et al 1995
[4] RVFL without bias and with direct link.
[5] RVFL with bias and with direct link. — Also original RVFL 1992, 1994
[6] Statistic value derived from Eq. (4). — Equation is 2 slides back

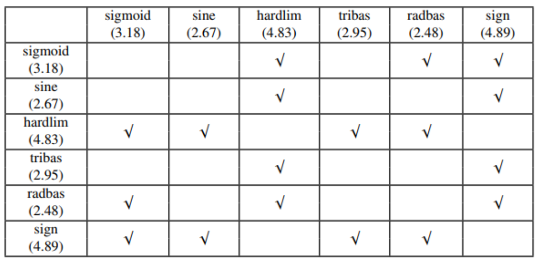
The direct links lead to better performance than those without in all cases.

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.

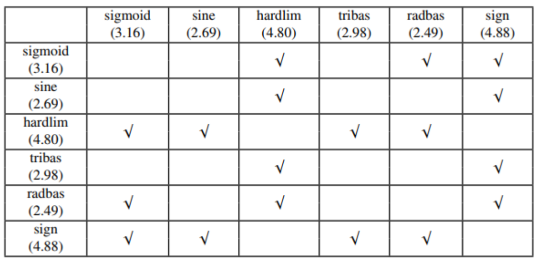
Statistical significance test for different activation functions. The values in bracket stands for their average rank. Lower values are for better performance. The mark √ indicates that these two methods are statistically significantly different.

(a) -bias, -link, Ridge regression



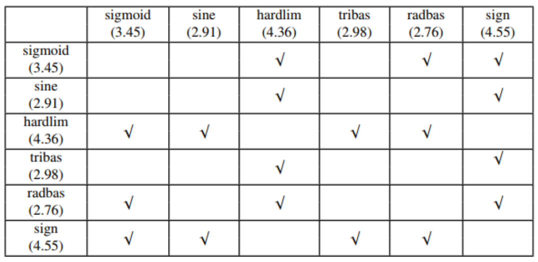
	sigmoid (3.18)	sine (2.67)	hardlim (4.83)	tribas (2.95)	radbas (2.48)	sign (4.89)
sigmoid (3.18)			√		√	√
sine (2.67)			√			√
hardlim (4.83)	√	√		√	√	
tribas (2.95)			√			√
radbas (2.48)	√		√			√
sign (4.89)	√	√		√	√	

(b) +bias, -link, Ridge regression



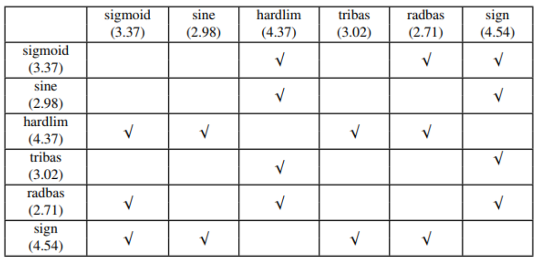
	sigmoid (3.16)	sine (2.69)	hardlim (4.80)	tribas (2.98)	radbas (2.49)	sign (4.88)
sigmoid (3.16)			√		√	√
sine (2.69)			√			√
hardlim (4.80)	√	√		√	√	
tribas (2.98)			√			√
radbas (2.49)	√		√			√
sign (4.88)	√	√		√	√	

(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)	√	√		√	√	

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,

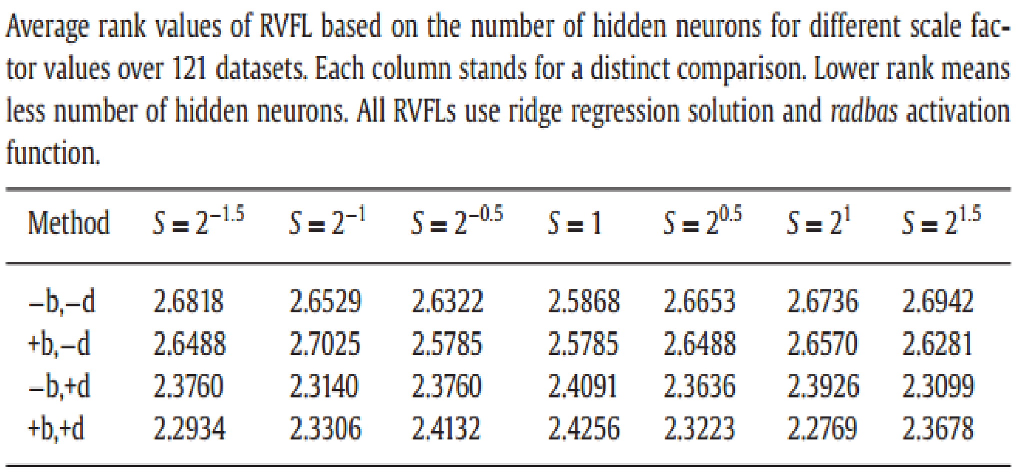
	Sigmoid	Sine	Hardlim	Tribas	Radbas	Sign
Ridge Regression	64	57	68	*76*	58	63
Moore–Penrose pseudoinverse	52	56	46	*42*	59	51

(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

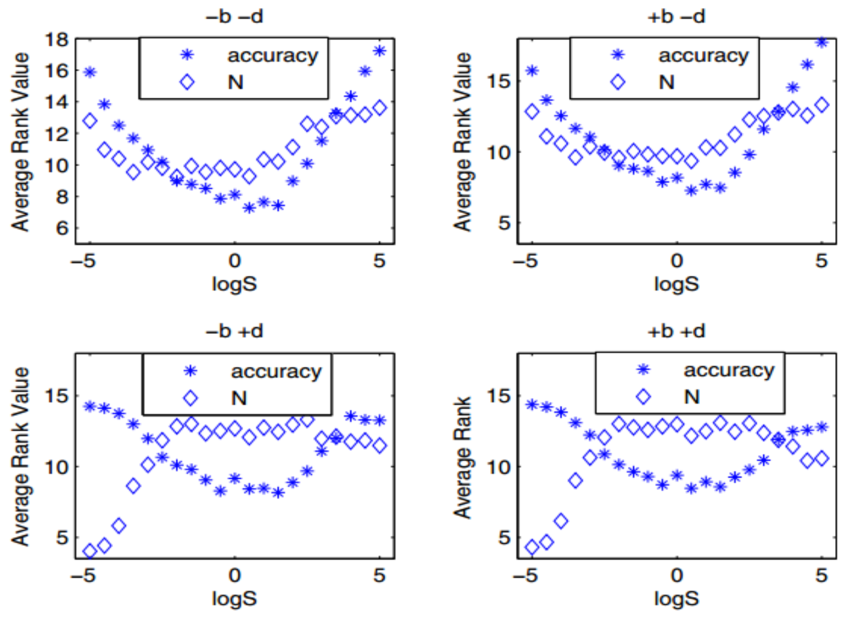
Results: Scaling the Randomization range of input weights and biases



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

Deep RVFL is largely insensitive to range scaling, i.e. 0-1 uniform randomization is sufficient.



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.

RVFL Vs Extreme Learning Machines

- The main difference between the RVFL and the ELM 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, if the input feature dimensionality is large. In this case, 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...
- 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.
- **Advantage of direct links is magnified in deep RVFL.**
- Evidences of RVFL's Superiority Over ELM in recent research publications.

Deep RVFL and Kernel Ridge Regression

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

https://www3.ntu.edu.sg/home/epnsugan/

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

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 ensemble 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

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:

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

←Concatenated data

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

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

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.(**Refer to "Learning in RVFL" slide)**

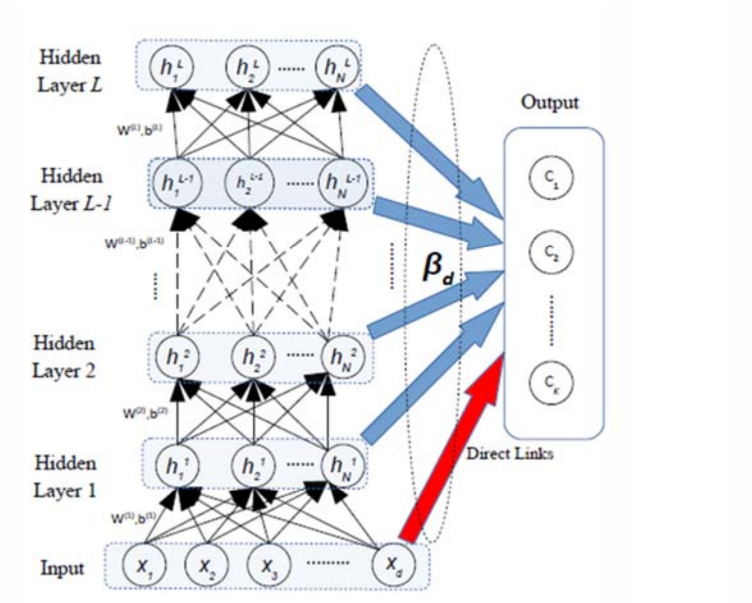


Figure 2: Framework of a dRVFL network. It consists of several hidden layers stacked on top of each other whose parameters (weights and biases between the hidden layers) are randomly generated and kept fixed during the training. Only the output weights β_d need to be computed as in the shallow RVFL network.

Single classifier, single classification decision.

Reference:

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

Ensemble Deep Random Vector Functional Link Network

- The ensemble deep RVFL network serves three purposes: 1) instead of using features **concatenated** from all layers, it employs rich intermediate features for final decision making **separately.** 2) the ensemble is obtained by a single dRVFL structure 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.

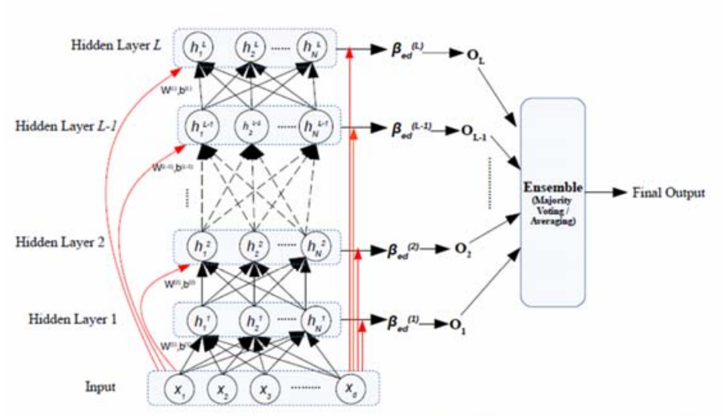


Figure 3: Framework of ensemble deep RVFL network (edRVFL). It differs from dRVFL in that the computation of final output weight β_d is decomposed into several small β_{ed}. Specifically, each small β_{ed} is independently computed (treated as independent model) and the final output is obtained by using 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 β_{ed} are then solved independently using primal or dual solutions.(← **Refer to "Learning in RVFL" slide)**

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.

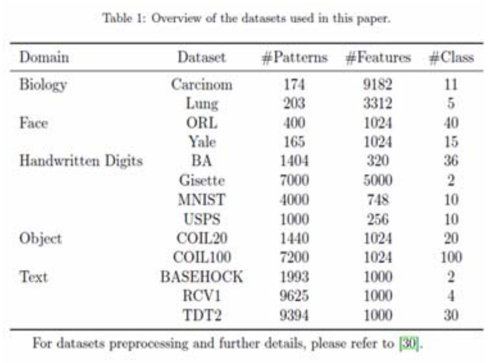


Table 1: Overview of the datasets used in this paper.

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].

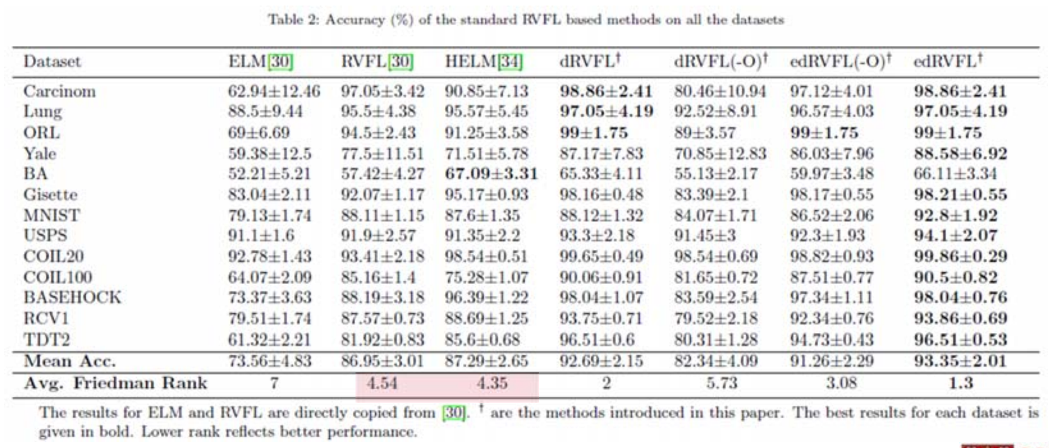


Table 2: Accuracy (%) of the standard RVFL based methods on all the datasets

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.

Summary

- Presented a deep learning model (dRVFL) based on random vector functional link network with the parameters of the hidden layers were randomly generated and kept fixed with the output weights computed using a closed form solution.
- The dRVFL network extracts feature representations through several hidden layers using 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, such as the SP-RVFL (2019).
- Extensive experiments on several classification datasets showed that the our deep learning RVFL networks achieve better generalization compared to pertinent randomized neural networks.

Limitations

- The major limitation is that ensemble deep RVFL does not perform convolutions.
- Therefore, its performance on image or sequential or temporal data is not competitive if the raw image or sequence data is used as inputs.
- If outputs of CNN (i.e. CNN extracted features) are used, the performance is very good.
- For tabular data, ensemble deep RVFL is very competitive.

Kernel Ridge Regression (KRR)

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:

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

Quadratic minimization Objective with kernel trick:

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

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

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 2000s or so.

NANYANG TECHNOLOGICAL UNIVERSITY

KRR Vs Kernel ELM

- Kernel ELM was derived in 2010 from the ELM proposed.
- 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.

NANYANG TECHNOLOGICAL UNIVERSITY

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 …

Random Forests

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

Some Software Resources Available from:

https://github.com/P-N-Suganthan

Ensemble of Classifiers

- A committee of classifiers trained either independently or sequentially.
- Obtained by perturbing the training set or injecting some randomness in each classifier and aggregating the outputs of the these classifiers in a suitable way.
- Decision trees (DT) and Neural networks (NN): commonly used classifiers for constructing ensembles.
- We require unstable or randomization-involved classifiers as opposed stable classifiers with unique global optimum solutions.

Decision Tree

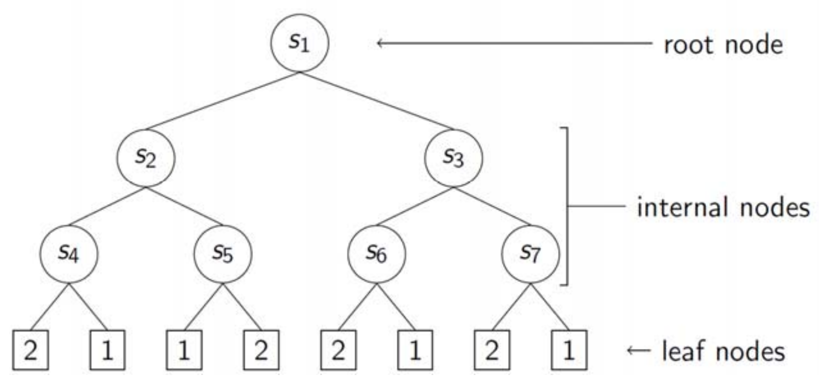
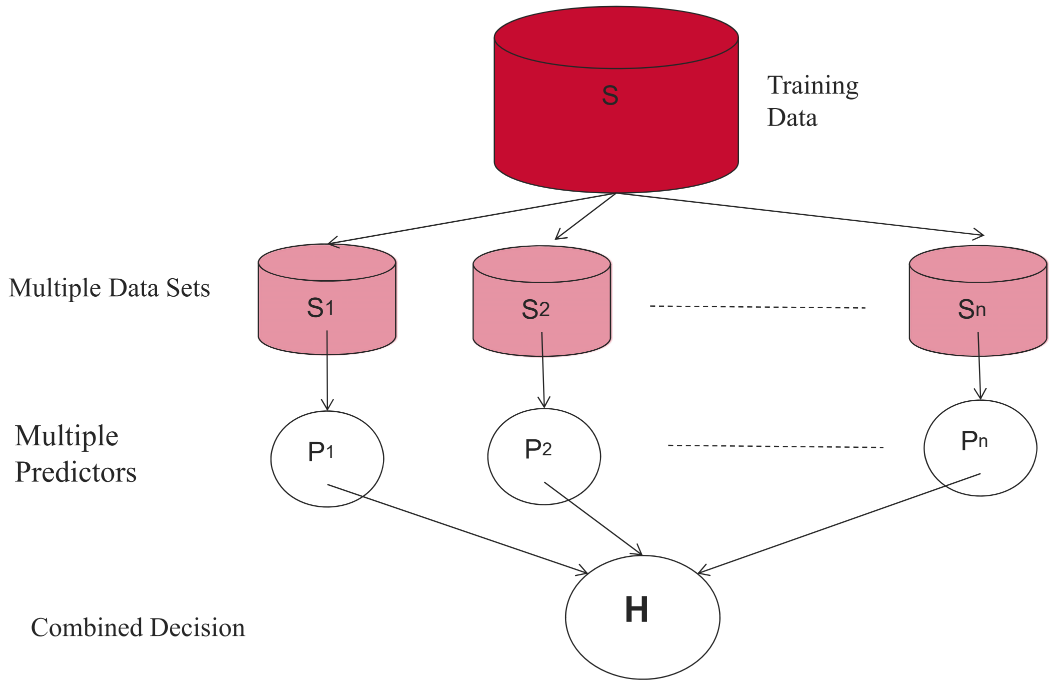


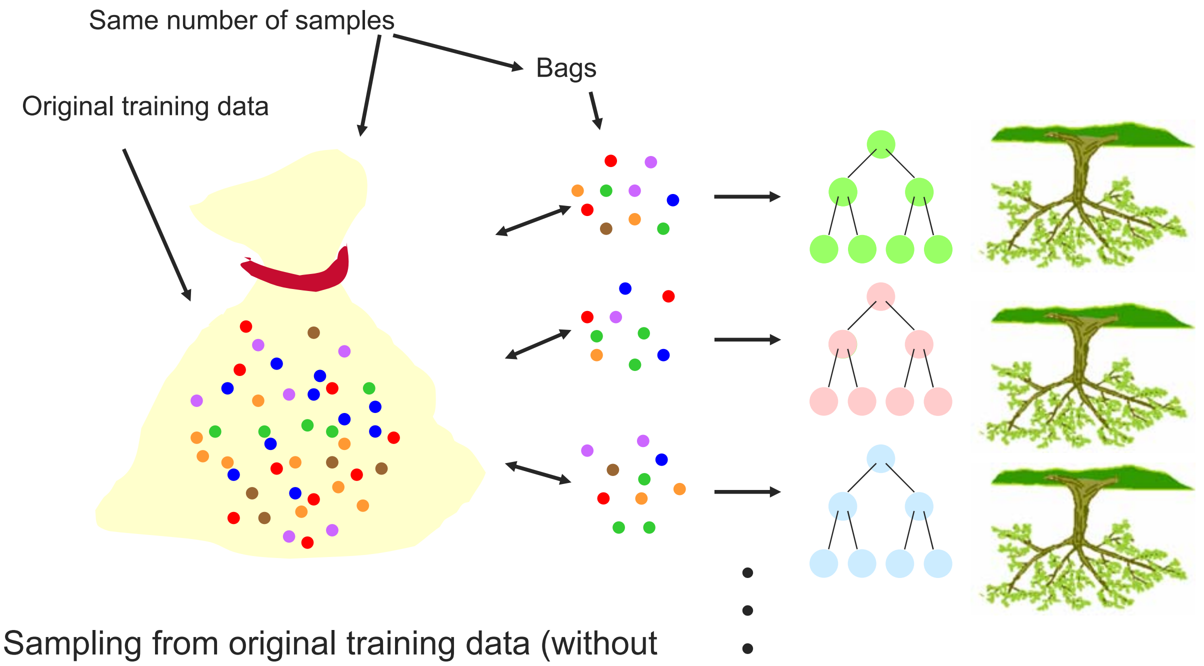
Figure: Decision Tree

- a popular machine learning algorithm
- employs recursive partitioning

General Overview



Bagging(1)



Sampling from original training data (without deletion) Usually to the same number of samples In each bag. Around 65% of distinct samples and 35% duplicates.

Bagging(2)

1. For i=1: No.Trs // No.Trs is number of trees or bags
 a) Draw (with replacement) from the training set to generate the training data S_i
 b) Learn a classifier C_i on S_i .

2. For each test example
 a) Try all classifiers C_i
 b) Predict the class that receives the highest number of votes.

Random Forests

Random Forest Algorithm:

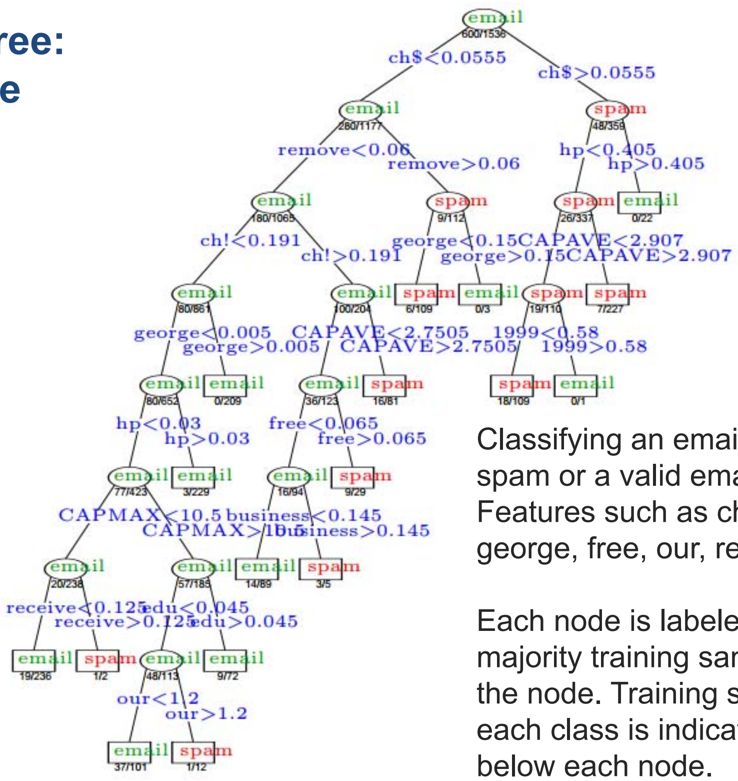
· Training phase:
Given:
$X := N \times M$ is the training dataset, where N is the number of the training data, M is the dimension of each data.
$Y := N \times 1$ is the labels of the training set.
L is the ensemble size, which means the number of trees in the forests.
T_i refers to each random tree in the forest, $i = 1...L$.
m is the number of features randomly selected to split in each non-leaf node.
$minleaf$ is the maximum number of samples in an impure terminal node.

1. Generate the training set for T_i by sampling N times from all N available training cases with replacement.
2. At each node the best split is calculated using the m randomly chosen features in the training set for T_i.
3. Go to Step 2 until T_i is fully grown until reach an pure node or the number of samples are smaller than "$minleaf$" .

· Classification phase:
For a given sample, it is pushed down each tree in the forests and each tree in the forests will give one vote on the predicted label of this sample. In this case, the predicted label of this sample is determined as the one which has the most votes in the forests.

L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5–32, 2001.

Decision Tree: An Example



Classifying an email as either as spam or a valid email using Features such as ch! , ch$, 1999, george, free, our, receive, …

Each node is labeled based on majority training samples reaching the node. Training samples of each class is indicated as x/y below each node.

Standard Decision Tree

- Each internal node is associated with a test/split function defined as:

$$f(\mathrm{x}, \theta) = \begin{cases} 1 \ \text{if}\ \mathrm{x}(\theta_1) < \theta_2 \\ 0 \ \text{otherwise} \end{cases}$$

where $\theta_1 \epsilon \{1,2, \dots d\}$ and $\theta_2 \epsilon \mathbb{R}$ is a threshold.

A node of decision Tree

Assume that n training samples (each with M features) reach an internal node.

Each row is a data point

$$\begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,d} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,d} \\ X_{3,1} & X_{3,2} & \cdots & X_{3,d} \\ \vdots & \vdots & \ddots & \cdots \\ X_{n-1,1} & X_{n-1,2} & \cdots & X_{n-1,d} \\ X_{n,1} & X_{n,2} & \cdots & X_{n,d} \end{bmatrix}$$

Threshold

How to optimize the threshold for each feature?

Each column is a feature. There are M features per training sample.

Optimize threshold for each feature

- the threshold should be optimized so that <span>Gini impurity</span> is maximized: $Gini_{beforesplit} - Gini_{aftersplit}$

$$Gini(t)_{beforesplit} = 1 - \sum_{i=1}^{c} \left(\frac{n_{w_i}}{n}\right)^2$$

Eqn (a)

$$Gini(t)_{aftersplit} = \frac{n^l}{n}\left[1 - \sum_{i=1}^{c} \left(\frac{n_{w_i}^l}{n^l}\right)^2\right] + \frac{n^r}{n}\left[1 - \sum_{i=1}^{c} \left(\frac{n_{w_i}^r}{n^r}\right)^2\right]$$

Eqn (b)

C -total number of classes at the node
n – total samples
r - right branch
l – left branch
wi – the same as *i*.

There are other impurity criteria such as information gain, etc.

Example

Imagine that 50 training samples belonging to 5 classes (10 samples each) reaching a node. Pay attention to the squared terms in the summation in the previous slide.

Before the split: **1-** $\sum_{i=1}^{5} \left(\frac{10}{50}\right)^2$

Imagine 2 hypothetical cases after split:
(a) 40 samples belonging to first 4 classes on the right and 10 samples belonging to the last class on the left.

After Split: $\frac{10}{50}\left[1 - \left(\frac{10}{10}\right)^2\right] + \frac{40}{50}\left[1 - \sum_{i=1}^{4} \left(\frac{10}{40}\right)^2\right]$

(b) 5 samples belonging to each class on the left and right leaf nodes.

After Split: $\frac{25}{50}\left[1 - \sum_{i=1}^{5} \left(\frac{5}{25}\right)^2\right] + \frac{25}{50}\left[1 - \sum_{i=1}^{5} \left(\frac{5}{25}\right)^2\right]$

In practice…

- The example in the previous slide considers only a single node.
- The tree in slide 8 has ~16 nodes.
- The Random Forest can have ~500 trees.
- The number of features will also increase this computation.
- Hence, the example in the previous slide is repeated numerous times …

Unsupervised Learning

- Clustering by Affinity Propagation method
- Stacked/Deep Auto-encoders

Unsupervised learning

- We have access to $\{x_1,x_2,x_3,..,x_N\}$ but not $\{y_1,y_2,y_3,..,y_N\}$ (i.e. data without labels)

- Potential Tasks/Objectives:

1. Clustering / grouping
2. Internal Representation Learning
3. Feature extraction
4. Discovering interesting trend
5. Data compression
6. ………

Affinity Propagation Clustering Algorithm

- Affinity Propagation(AP) is a centroid based clustering algorithm, which does not require specification of the number of clusters a priori
- AP has a centroid. All data points in the sample may become the centroid in the AP algorithm, which is called ‘Exemplar’, instead of the cluster centers obtained by averaging multiple points (like K-Means)

Pros

- There is no requirement for the symmetry of the distance matrix. The AP starts the algorithm by entering the similarity matrix, so the data is allowed to be asymmetric, and the data is applicable in a very large range.
- AP is not sensitive to initial values. If AP clustering algorithm is executed multiple times, the results will be the same.

Cons

- The complexity of the algorithm is high, which is $O(N^2 \log N)$ and K-Means is only $O(N \times k)$ complexity. Therefore, when N is large (N> 3000), the AP clustering algorithm often takes a long time.
- The AP algorithm needs to calculate the similarity between each pair of data samples in advance

The concepts

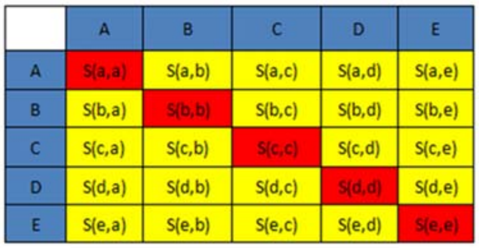
- The 'Exemplar' is the center of the cluster
- The 'Similarity' $s(i, k)$ indicates how well the data point with index k is suited to be the exemplar for data point i.
- The real number 'Preferences' p . The number of identified exemplars is influenced by the values of the input preferences.

The concepts

- The 'Responsibility' $r(i, k)$, sent from data point i to candidate exemplar point k, reflects the accumulated evidence for how well-suited point k is to serve as the exemplar for point i, taking into account other potential exemplars for point i.
- The 'Availability' $a(i, k)$, sent from candidate exemplar point k to point i, reflects the accumulated evidence for how appropriate it would be for point i to choose point k as its exemplar, taking into account the support from other points for which point k should be an exemplar

AP-Step 1: Get the Similarity Matrix and Set the Preference

- The similarity s can be set to a negative squared error (Euclidean distance): For points x_i and x_k, $s(i, k) = -\|x_i - x_k\|^2$.



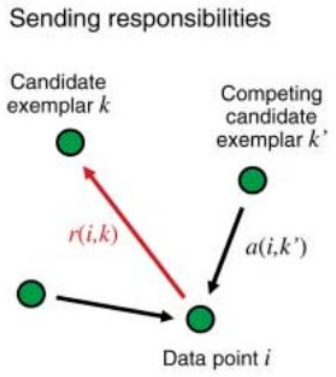
	A	B	C	D	E
A	S(a,a)	S(a,b)	S(a,c)	S(a,d)	S(a,e)
B	S(b,a)	S(b,b)	S(b,c)	S(b,d)	S(b,e)
C	S(c,a)	S(c,b)	S(c,c)	S(c,d)	S(c,e)
D	S(d,a)	S(d,b)	S(d,c)	S(d,d)	S(d,e)
E	S(e,a)	S(e,b)	S(e,c)	S(e,d)	S(e,e)

- Normally, the value of p could be the median of the input similarities or their minimum

AP-Step 2: Calculate and Update the Responsibility Matrix

- To begin with, the availabilities are initialized to 0: $a(i,k) = 0$. Then the responsibilities are computed according to Equation (1):

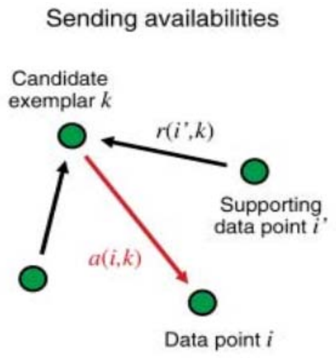
$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\} \quad \text{— (1)}$$



AP-Step 3: Calculate and Update the Availability Matrix

- The availability update gathers evidence from data points as to whether each candidate exemplar would make a good exemplar

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ s.t. } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\} \quad \text{— (2)}$$



AP-Step 4: Choose the exemplar

- For point i, the value of k that maximizes $a(i,k) + r(i,k)$ either identifies point i as an exemplar if $\mathrm{k} = i$, or identifies the data point that is the exemplar for point i
- When updating the messages, they are need to be damped to avoid numerical oscillations. Each message is set to λ times its value from the previous iteration plus $1 - \lambda$ times its prescribed updated value

$$r(i,k) = (1-\lambda)r(i,k) + \lambda r(i,k)'$$
$$a(i,k) = (1-\lambda)a(i,k) + \lambda a(i,k)'$$

Pseudocode of AP

```
Initialize availabilities $\mathrm{a}(\mathrm{i},\mathrm{k}) = 0$
**do**{
        Update, using Equation (1), all the responsibilities given the availabilities
        Update, using Equation (2), all the availabilities given the responsibilities
        Combine availabilities and responsibilities to obtain the exemplar decisions
} **until** Termination criterion is met
```

An Analogy

- An analogy can be used to understand these two quantities and the process of alternation between them: election
- Everyone participates in the election (everyone is an elector and a candidate), and several are elected as representatives
- $s(i,k)$ is equivalent to an inherent preference of i for the person who chooses k
- $r(i,k)$ represents the score of the strongest competitor minus $s(i,k)$,which can be understood as the degree of advantage of k in the competition against i

The Analogy

- The update process of $r(i,k)$ corresponds to the selection of each candidate of elector i (the more outstanding the more attractive)
- $a(i,k)$: As can be seen from the formula, all values of $r(i',k) > 0$ have a positive addition to a. Corresponding to our metaphor, it is equivalent to the voter i seeing through an online poll about k: Many people (voter i') think that k is good ($r(i',k) > 0$), then the voter i correspondingly feels that k is good and is a trustworthy choice.

The Analogy

- The update process of $a(i,k)$ corresponds to the impact of poll k on candidate i on voter i (people who already have a lot of followers are more attractive)
- The process of alternation between the two can be understood as that the voters constantly compare and refer to the polls given by each candidate.
- The idea of $r(i,k)$ reflects competition, while $a(i,k)$ is to make clustering more successful.

Auto-encoders

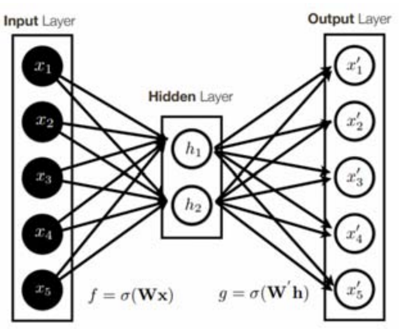
Autoencoders

- Auto = self ; Autoencoders = encoders + decoders
- Feed-forward network desired to reproduce the input

  Encoder: $f = \sigma(\mathbf{Wx})$

  Decoder: $f = \sigma(\mathbf{W'h})$
- Reproduced input $\mathbf{x}' = g(f(\mathbf{x}))$
- Objective: Minimize loss $\mathcal{L}(\mathbf{x}', \mathbf{x})$
- Optimization problem: $\arg\min_{\Theta,\Theta'} \mathbb{E}(L(\mathbf{X}, g(f(\mathbf{X}))))$

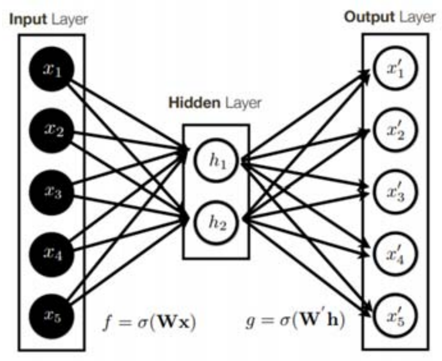
  where Ө,Ө’ are the parameters of encoder and decoder respectively.



- For linear reconstruction,
  - $\mathcal{L}(x, y) = ||x - y||^2$

Autoencoders

- If the number of hidden neurons is **smaller** than the input dimension,
  - data compression: reconstruction of the data far from the training distribution will be difficult.
- If the number of hidden neurons is **larger** than the input dimension,
  - no data compression
  - can learn to simply copy, without learning of meaningful features
  - regularization needed

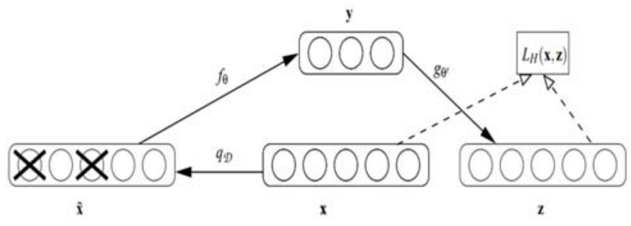


De-noising Autoencoders

Add noise to the input but learn to re-construct the original

- Prevents copying
- Learn better (robust) features
- Gaussian noise, binary masking (a fraction of the elements of **x** is forced to 0), salt and pepper noise (a fraction of the elements of **x** is set to their minimum or maximum), etc.

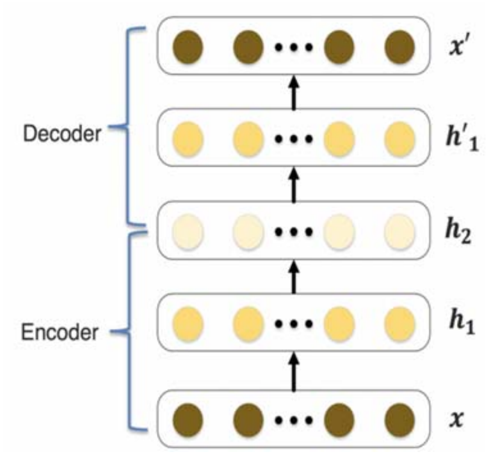
- Optimization problem: $\arg\min_{\Theta,\Theta'} \mathbb{E}(L(\mathbf{X}, g(f(\tilde{\mathbf{X}}))))$



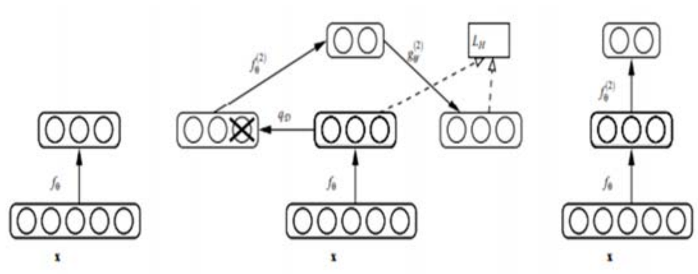
The denoising autoencoder architecture. An example $\mathbf{x}$ is stochastically corrupted (via $q_\mathcal{D}$) to $\tilde{\mathbf{x}}$. The autoencoder then maps it to $\mathbf{y}$ (via encoder f_θ) and attempts to reconstruct $\mathbf{x}$ via decoder $g_{\theta'}$, producing reconstruction $\mathbf{z}$. Reconstruction error is measured by loss $L_H(\mathbf{x},\mathbf{z})$.

Stacked (deep) Autoencoders and Denoising Autoencoders

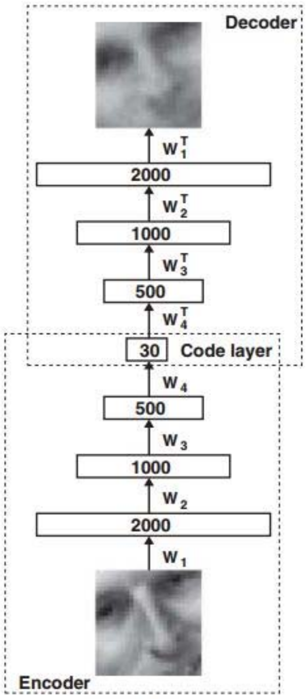
- Good for compression, an internal representation for the data.
- Fine-tune the pre-trained autoencoder for a task
  - i.e. transfer learning scenario



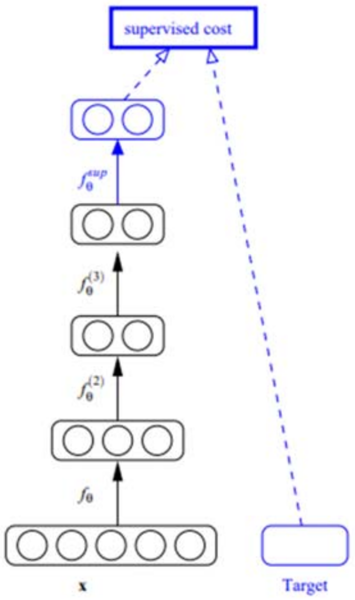
Stacked (deep) Autoencoders and Denoising Autoencoders



Stacking denoising autoencoders. After training a first level denoising autoencoder its learnt encoding function f_θ is used on clean input (left). The resulting representation is used to train a second level denoising autoencoder (middle) to learn a second level encoding function $f_\theta^{(2)}$. From there, the procedure can be repeated (right).



Stacked (deep) Autoencoders and Denoising Autoencoders

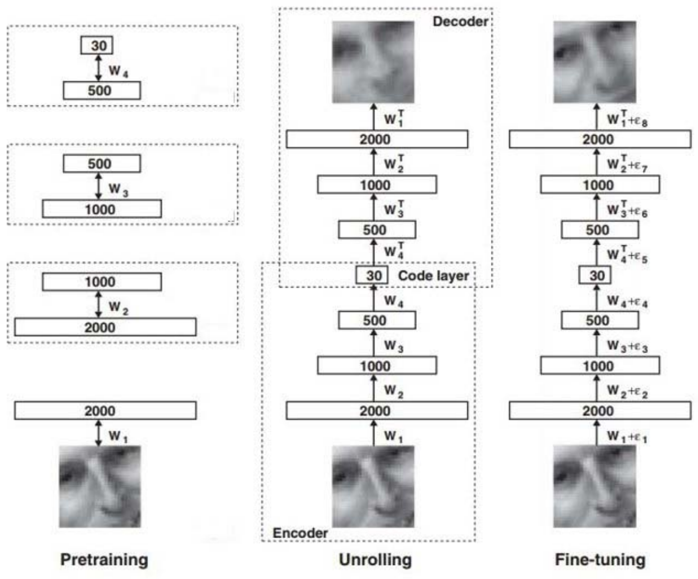


If we have large number of unlabelled data and a small number of labelled data, this approach is very useful.

Another scenario is transfer learning which is commonly used in image recognition tasks.

Fine-tuning of a deep network for classification. After training a stack of encoders as explained in the previous figure, an output layer is added on top of the stack. The parameters of the whole system are fine-tuned to minimize the error in predicting the supervised target (e.g., class)

Training Deep Autoencoders



Pretraining consists of learning a stack of autoencoder, each having only one layer of feature detectors. The learned feature activations of one autoencoder are used as the "data" for training the next autoencoder in the stack. After the pre-training, the autoencoders are "unrolled" to create a deep autoencoder, which is then fine-tuned using back-propagation of error derivatives. In randomized neural nets, closed form solution is employed.