Kernelization of ensemble Deep Random Vector Functional-Link Networks

Vihaan Akshaay Rajendiran

May 2, 2025

Abstract

This paper presents a study on Kernelized ensemble deep RVFL (K-eDRVFL) networks, a novel approach for machine learning tasks. We introduce the theoretical foundation of K-eDRVFL, discuss its architecture, training process, and ensemble techniques. We further extend this algorithm to use knowledge from the ensemble deep RVFL layers, by combining both and defining 'Mixed Kernelized ensemble Deep Random Vector Functional-Link Network'. Additionally, we provide experimental results to demonstrate its performance compared to other state-of-the-art methods.

1 Introduction

Artificial Intelligence (AI) is changing the way we approach traditional tasks, handling challenges that were once thought to be solely the realm of human thinking, especially in making decisions and solving complex problems. Recently, it can be noted that AI is significantly altering approaches and procedures, aiding humans in addressing challenges across various sectors, including Healthcare, Finance, Transportation, Education, Entertainment, Agriculture, Manufacturing and many more. At the heart of AI's evolution, deep learning has consistently been a fundamental framework.

Deep learning models, a subset of AI, utilize neural networks designed to mimic the human brain's structure and function. Deep Neural Networks, delve deeper into data representation than traditional algorithms, recognizing intricate patterns and complexities. They are modeled Their architectural design

In parallel, there has also been a lot of progress in another class of neural networks that use randomization based techniques with closed-form solution to overcome the issues that accompany BP based training. Following the universal approximation theorem, the architecture of single hidden layer feed-forward neural networks (SLFN) has been a subject of extensive research and investigation in the past. RVFL is an efficient learning method that can be adopted to train SLFNs. RVFL has quickly gained substantial recognition due to its outstanding performance across a wide range of diverse domains despite having a simple architecture and fast training speed. In our work, we compile existing kernel versions of RVFLs and extend it to the ensemble deep random vector functional link neural network formulation (eDRVFL) and present the theoretical formulation for Kernelized ensemble Deep Random Vector Functional-Link Network (K-edRVFL). In addition, we try to extend the meta model to utilize both kernel and non-kernel versions together allowing the agent to either choose or use both the versions better mode to make a prediction.

Specifically, we propose 4 variations/models as follows:

- K-eDRVFL: Each layer of the eDRVFL is replaced with the kernel version and the final prediction is made as an ensemble of predictions from each layer.
- Switch K-eDRVFL:
- Mix K-eDRVFL:
- Ensemble Kernel RVFL: We consider the Kernel RVFL formulation (cite here from related work) but add multiple layers, each performing kernel operations independently from the input layer and the prediction is made as an ensemble.

2 Related Work

2.1 Random Vector Functional Link Network (RVFL)

Write some theory on kernel(!!!)how

The optimization problem of RVFL with regularization can be defined as [Mal+23]

$$\min_{\beta} ||D\beta - Y||^2 + C||\beta||^2$$

where $D = [H \quad X]$ is the concatenation matrix containing input data (X) and outputs from the hidden layer (H) and β is the output weight matrix.

The solution to this optimization problem is:
primal space: $\beta = (D^T D + \lambda I)^{-1} D^T Y$
dual space: $\beta = D^T (D D^T + \lambda I)^{-1} Y$
where I is an identity matrix of appropriate dimension.

2.2 Deep Random Vector Functional Link Network (dRVFL)

This is an extension of RVFL with L hidden layers, and each of those layers act as as an independent RVFL. The hidden values are calculated as below:

$H^{(1)} = g(X * W^{(1)})$ and $H^{(k)} = g(H^{(k-1)} * W^{(k)})$

The augmented matrix is:
$D = [H^{(1)}\ H^{(2)}\ \ldots\ H^{(L-1)}\ H^{(L)}\ X]$

and the prediction/output of the network becomes:
$Y = D\beta_d$

The optimization function is very similar to that of RVFL

$$\min_{\beta_d} ||D\beta_d - Y||^2 + C||\beta_d||^2$$

The solution to the same is:
primal space: $\beta_d = (D^T D + \lambda I)^{-1} D^T Y$
dual space: $\beta_d = D^T (DD^T + \lambda I)^{-1} Y$

We choose either of the solutions based on the smaller matrix inverse execution.

2.3 Ensemble Deep Random Vector Functional Link network

Let's say that the value of the nodes in the k'th layer is given by H^k. We treat each layer as a separate RVFL model and the corresponding X^k and H^k are as follows. If X is the input data,

$X^k = [H^{k+1} \quad X]$
$H^k = g(X^k * W^k)$

Where W^k refers to the weights connecting hidden layers k-1 and k. Now we concatenate with the input features to make the prediction.

For each model, let's define D as, $D^k = [H^k \quad X^k]$ and
$Y_{prediction} = D^k * \beta^k$

Similar to the RVFL formulation, the solutions are as follows:
primal space: $\beta_k = (D^T D + \lambda I)^{-1} D^T Y$
dual space: $\beta_k = D^T (DD^T + \lambda I)^{-1} Y$

2.4 Kernel-based Random Vector Functional Link Network (K-RVFL) [!Check with original paper once]

Instead of having a hidden layer whose values are obtained by random weights, this method uses a non-linear transformation (like the gaussian kernels) for obtaining values in place of the hidden layer. Therefore, the augmented matrix becomes,

$D = [\phi(X) \quad X]$

Now we refer to the dual solution:

$$\beta = D^T (DD^T + \lambda I)^{-1} Y = [\phi(X) \quad X]^T [\phi\phi^T + XX^T + \lambda I]^{-1} Y$$

Now, if we wish to predict for some new data x_{new},
$Y_{x_{new}} = [\phi(x_{new}) \ (x_{new})]\beta = [\phi(x_{new}) \ (x_{new})][\phi(X) \ X]^T [\phi\phi^T + XX^T + \lambda I]^{-1} Y$

Following the kernel matrix entries,

$$Y^k_{x_{new}} = [x_{new} X^T + k(x^k, X^k)][\phi\phi^T + XX^T + \lambda I]^{-1} Y$$

2.5 Kernelized Random Vector Functional Link Network(RVFL$_{ker}$)

Let $\phi(x)$ be a feature mapping function. Now the redefined matrix D is as follows:
$D = [H \ \ \phi(X)]$

Now we refer to the dual solution:

$$\beta = D^T(DD^T + \lambda I)^{-1}Y = [H \ \ \phi]^T[HH^T + \phi\phi^T + \lambda I]^{-1}Y$$

Now, if we wish to predict for some new data x_{new},
$Y_{x_{new}} = [h(x_{new}) \ \phi(x_{new})]\beta = [h(x_{new}) \ \phi(x_{new})][H \ \phi]^T[HH^T + \phi\phi^T + \lambda I]^{-1}Y$

Here, we apply the kernel trick. We define the kernel function (Macer's kernel matrix) as follows:

$$\begin{pmatrix} k(a_2, b_1) & \dots & k(a_2, b_m) \\ \vdots & \ddots & \vdots \\ k(a_n, b_1) & \dots & k(a_n, b_m) \end{pmatrix}$$

Therefore, using these kernel matrices in the solution, we obtain

$$Y^k_{x_{new}} = [h(x_{new})H^T + k(x^k, X^k)][HH^T + \phi\phi^T + \lambda I]^{-1}Y$$

3 Our Proposed Frameworks

This is just a rough draft to help with implementation. I will refine things later once we start getting results.

3.1 Ker-eDRVFL [Kernel ensemble Deep RVFL]

In our first proposed model (i.e. K-eDRVFL) at each layer, we use the concatenation of hidden layer and the feature mapping of the current layer's X (i.e. input data and hidden data of previous layer concatenated) to explore kernelisation.

3.1.1 Mathematical Formulation:

Equations for the k'th model:
$X^k = [H^{k-1} \ \ X]$ and $H^k = g(X^kW^k)$
And finally,

$$D^k = [H^k \quad \phi(X^k)]$$

Now we refer to the dual solution:

$$\beta^k = D^T(DD^T + \lambda I)^{-1}Y = [H \ \ \phi]^T[HH^T + \phi\phi^T + \lambda I]^{-1}Y$$

where $H = H^k$ and $\phi = \phi(X^k)$
Now, if we wish to predict for some new data x_{new},

$$Y_{x_{new}} = [h(x_{new})\ \phi(x_{new})]\beta^k = [h(x_{new})\ \phi(x_{new})][H\ \phi]^T[HH^T + \phi\phi^T + \lambda I]^{-1}Y$$

Here, we apply the kernel trick. We define the kernel function (Macer's kernel matrix) as follows:

$$\begin{pmatrix} k(a_2,b_1) & \dots & k(a_2,b_m) \\ \vdots & \ddots & \vdots \\ k(a_n,b_1) & \dots & k(a_n,b_m) \end{pmatrix}$$

Therefore, using these kernel matrices in the solution, we obtain

$$Y^k_{x_{new}} = [h(x_{new})H^T + k(x^k_{new}, X^k)][HH^T + \phi\phi^T + \lambda I]^{-1}Y$$

where,
Y - data train Y labels
X - data train X
x^k - new data (concatenated) when in k'th layer
H^k - hidden layer node values for k'th layer/model obtained using original train data
x_{new} - new data to predict
h(x) - hidden layer values of k'th model from new data

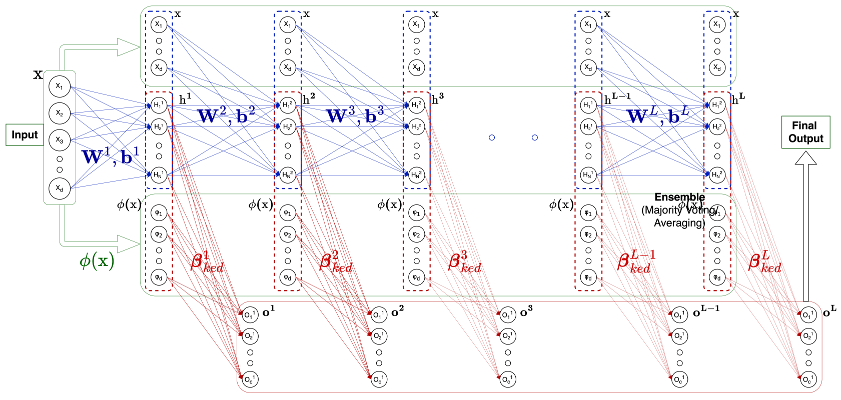


Figure 1: K-eDRVFL: Rough figure for reference

3.1.2 Implementation Details:

We split the prediction at each layer (k) into parts that involve x_{new} (which is the data we make prediction for) and that don't.

$$Y^k_{x_{new}} = \left[h(x_{new})H^T + k(x^k_{new}, X^k)\right]\left[HH^T + \phi\phi^T + \lambda I\right]^{-1}Y$$

We make this split since, we can save all that doesn't depend on the prediction input during the training phase and calculate the rest during prediction phase. We also store the hidden values of each layer obtained using the original train data X since it is used in the prediction later. Analogous to the notation of β for the weights saved during the training phase of eDRVFL, in our work, we call this part β_{semi}.

Therefore, now training the model essentially means storing β_{semi} at each layer,

$$\beta^k_{semi} = \left[HH^T + \phi\phi^T + \lambda I\right]^{-1} Y$$

and prediction becomes:

$$Y^k_{x_{new}} = \left[h(x_{new})(H^k)^T + k(x^k_{new}, X^k)\right]\beta^k_{semi}$$

3.2 Switch K-eDRVFL

Here, the key idea is to use either the kernel or the non-kernel version at each layer for the evaluation. This choice becomes a hyper-parameter tuned in the training phase.

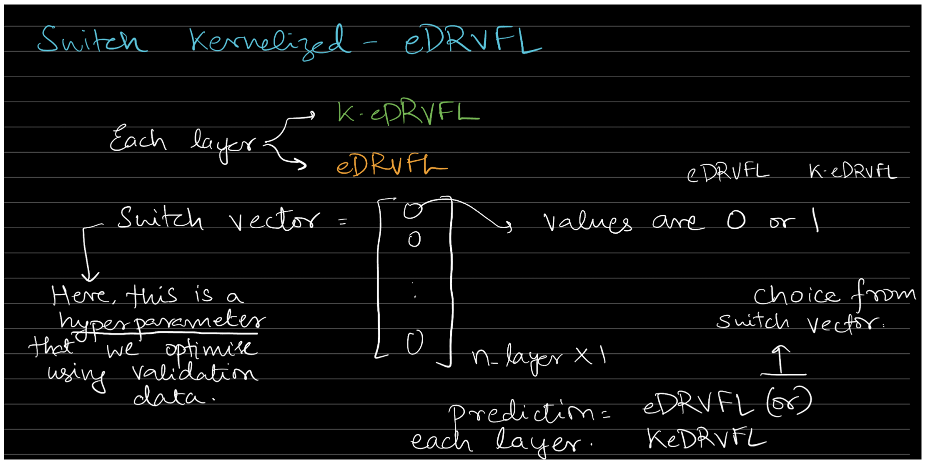


Figure 2: Switch Ker-eDRVFL: Rough figure for reference

3.2.1 Mathematical Formulation:

Similar to eDRVFL & K-eDRVFL, we do the exact same steps at each layer during the training phase by considering both possible configurations:

$$\beta_k = [H^k \quad X^k]^T (H^k (H^k)^T + X^k (X^k)^T + \lambda I)^{-1} Y$$

$$\beta_{semi}^k = \left[HH^T + \phi\phi^T + \lambda I\right]^{-1} Y$$

and during prediction phase, we choose which solution to use (either kernel or non-kernel eDRVFL prediction) at each layer based off of another hyperparameter (that is tuned) called the switch factor.

Switch vector is a vector with the size of number of layers and each entry of the vector is either a 0 or a 1 to denote eDRVFL or K-eDRVFL at each layer. Let the switch value for the kth layer be $sv^k \in \{0, 1\}$. Then prediction of kth layer is:

$$Y_{x_{new}}^k = (sv^k) * \left[h(x_{new})H^T + k(x_{new}^k, X^k)\right]\beta_{semi}^k + (1 - sv^k) * [H^k \quad X^k]\beta_k$$

3.2.2 Implementation Details:

We start by having a list to store β's and β_{semi}'s for each layer (called e_betas = [] and k_betas = []) in addition to storing H at each layer.. Instead of calculating weights for both configurations in each layer, we check if the corresponding switch value is 0 or 1 and depending on that store β or β_{semi} at each layer and store dummy data for the other component for that layer.

if $sv^k = 0$ then e_betas.append(β^k) and k_betas.append(None)
else if $sv^k = 1$ then e_betas.append(None) and k_betas.append(β_{semi}^k)

During prediction, we similarly check sv and decide to either use eDRVFL or KeDRVFL predictions in each layer.

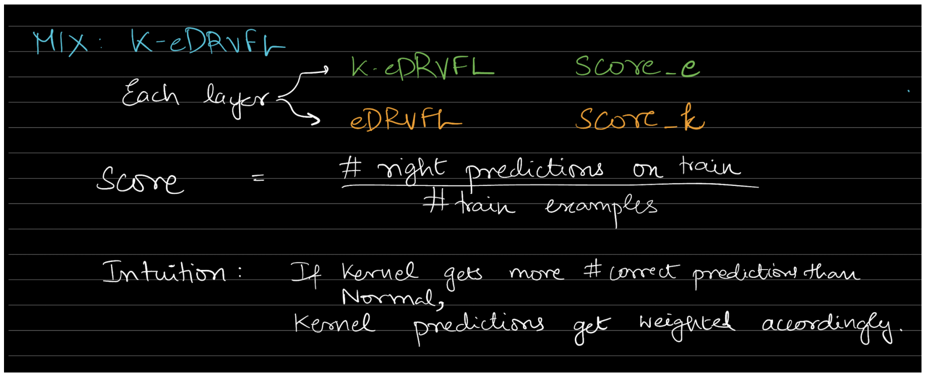
if $sv^k = 0$ then β^k = e_betas[k] and prediction $= [H^k \quad X^k]\beta_k$ else if $sv^k = 1$ then
β_{semi}^k = k_betas[k] and prediction $= Y_{x_{new}}^k = \left[h(x_{new})(H^k)^T + k(x_{new}^k, X^k)\right]\beta_{semi}^k$

3.3 Mix Kernel ensemble Deep RVFL

In this formulation, the key idea is to use both kernel and non-kernel version at each layer. The combined final output of each layer will be the predictions of a kernel and a non-kernel version and we take a weighted average. We learn the weights from the validation phase.

3.3.1 Mathematical Formulation:

In each layer, after we train both kernel and non-kernel versions of the model, we evaluate on the very same data it trained on to see which version (kernel or non-kernel) fit the data by calculating score_e and score_k:



MIX: K-eDRVFL

Each layer → K-eDRVFL Score_e

Each layer → eDRVFL Score_k

Score = # right predictions on train / # train examples

Intuition: If Kernel gets more # correct predictions than Normal, Kernel predictions get weighted accordingly.

Figure 3: Mix Ker-eDRVFL: Rough figure for reference

$$Score\_e = \frac{\text{\#right predictions by eDRVFL configuration on train}}{\text{\#total train examples}}$$

$$Score\_k = \frac{\text{\#right predictions by K-eDRVFL configuration on train}}{\text{\#total train examples}}$$

We then decide the output in each layer as a weighted average of outputs from both eDRVFL and K-eDRVFL configurations.

$$output\_net = \frac{(Score\_e * outputs\_e + Score\_k * outputs\_k)}{(Score\_e + Score\_k)}$$

3.3.2 Implementation Details:

We follow a setup similar to Switch K-eDRVFL and store βs and β_{semi} for every layer during training and during prediction calculate outputs from both kernel and non-kernel versions and predict combined output by doing a weighted calculation similar to what's mentioned above.

3.4 Ensemble Kernel RVFL

This is an extension of (cite K-RVFL). Kernel RVFL (K-RVFL) [Check section 2.4] uses kernel formulation to obtain the hidden layer representation and then solves for the final output.

Here in our extension, we wish to add the ensemble formulation by adding multiple layers and using different kernel functions at each layer.

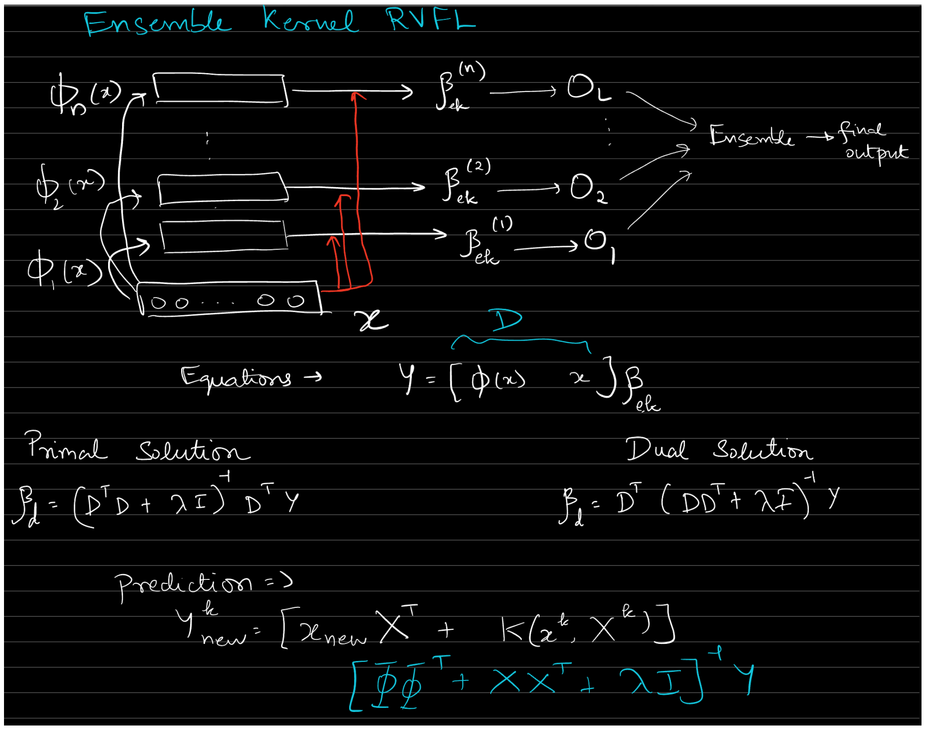

Ensemble Kernel RVFL
$\phi_n(x)$
$\phi_2(x)$
$\phi_1(x)$
$\beta_{ek}^{(n)}$
$\beta_{ek}^{(2)}$
$\beta_{ek}^{(1)}$
O_L
O_2
O_1
Ensemble → final output
x
Equations → $Y = \overbrace{[\phi(x) \quad x]}^{D} \beta_{ek}$
Primal Solution
$\beta_d = (D^T D + \lambda I)^{-1} D^T Y$
Dual Solution
$\beta_d = D^T (DD^T + \lambda I)^{-1} Y$
Prediction =>
$Y^k_{new} = [x_{new} X^T + K(x^k, X^k)]$
$[\Phi\Phi^T + XX^T + \lambda I]^{-1} Y$

Figure 4: eKRVFL: Rough figure for reference

3.4.1 Mathematical Formulation:

We have a different kernel function for each layer and a scalar called active value (av) for each of the layer depicting if we use this layer or not towards the final model's ensemble prediction. When av = 0, we ignore this layer and when av = 1, we use it.

Following equations similar to K-RVFL, we follow a similar setup for eK-RVFL. The partial semi-beta we save here for each layer, is conveniently called β^k_{esemi}

During the training phase,for each layer we calculate

$$\beta_{esemi} = [\Phi(X_{data})\Phi(X_{data})^T + X_{data}X_{data}^T + \lambda I]^{-1}Y$$

During prediction phase for each layer

$$Prediction = [K(x, X) + xX^T]\beta_{esemi}$$

3.4.2 Implementation Details:

The test on are (namely lin: [c], poly: [d, c], rbf: [sigma], sigmoid: [alpha, c],lap: [sigma], rat_quad: [c], multiquad: [c],cos: [bool], chisqr: [bool], hist: [bool])

and maintain a vector called active_vector which will be a vector of size 10 with values either 0 or 1 denoting use of that layer or not towards the ensemble of models that make prediction.

We start with semi_beta = [] to hold all the β_{esemi} values during training and if a layer is not active, we append None.

During training: if av = True: store

$$\beta_{esemi} = [\Phi(X_{data})\Phi(X_{data})^T + X_{data}X_{data}^T + \lambda I]^{-1}Y$$

if av = False: store None

During evaluation: if av = True:

$$Prediction = [K(x, X) + xX^T]\beta_{esemi}$$

if av = False: continue to next layer

4 Experiments

Provide information about the datasets used for evaluation, the preprocessing steps, and the metrics employed to assess the performance of K-eDRVFL networks.

4.1 Datasets (Text used from SNN paper need to be refined/changed)

121 UCI Machine Learning Repository datasets. The benchmark comprises 121 classification datasets from the UCI Machine Learning repository. We use the same split as the original SNN paper.

4.2 Compared Methods

4.3 Experimental settings

Each compared FNN method was optimized with respect to its architecture and hyperparameters on a validation set that was then removed from the subsequent analysis. The selected hyperparameters served to evaluate the methods in terms of accuracy on the pre-defined test sets. For the UCI data sets, the best hyperparameter setting was determined by a grid-search over all hyperparameter combinations using 15% of the training data as validation set

4.4 Experimental results on the UCI datasets

5 Results and Analysis

Present the results of the experiments and compare the performance of K-eDRVFL networks with other relevant models. Include visualizations and analyses to support your findings.

6 Conclusion

Conclude the paper by summarizing the key points discussed and emphasizing the contributions and significance of K-eDRVFL networks in the context of machine learning.

Acknowledgments

Thank any individuals or institutions that provided support, guidance, or assistance during the research.

References

[Mal+23] A.K. Malik et al. "Random vector functional link network: Recent developments, applications, and future directions". In: *Applied Soft Computing* 143 (Aug. 2023), p. 110377. ISSN: 1568-4946. DOI: `10.1016/j.asoc.2023.110377`. URL: `http://dx.doi.org/10.1016/j.asoc.2023.110377`.