Computational Intelligence

Project 4 – RBF NN

Vellios George Serafeim
velliosgeo@gmail.com

Vellios Georgios Serafeim

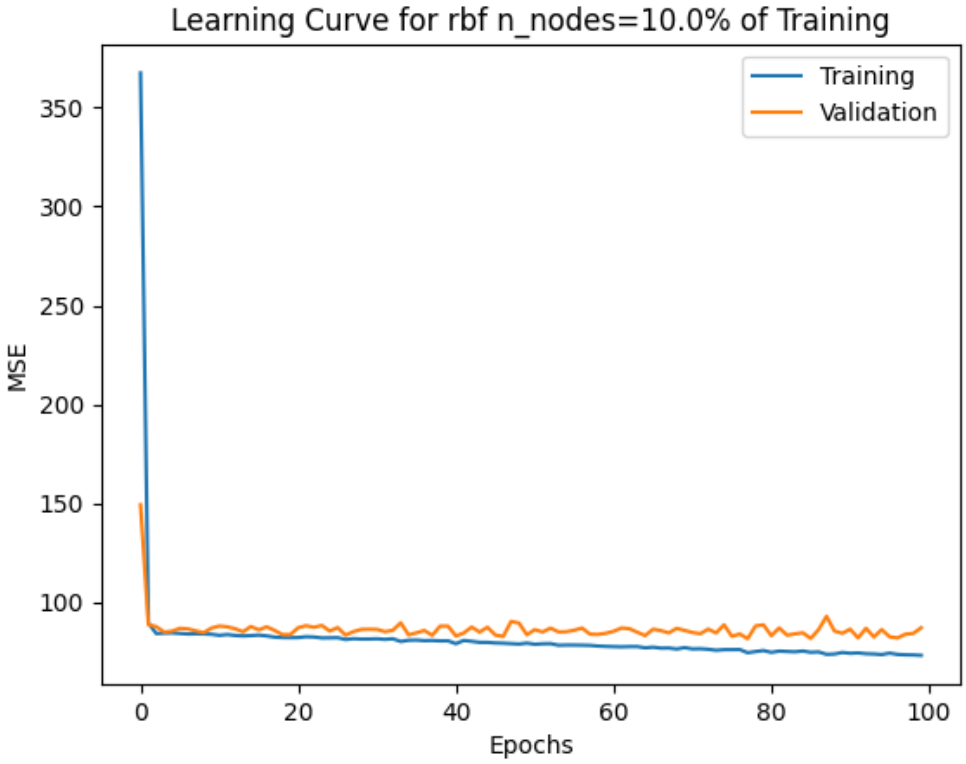
Solving a regression problem with an RBF network

The Boston housing Dataset containing 506 samples with 14 features is used for the regression problem. The keras library was used for the creation of the neural network. The RBF layer used is a variant of the layer created by *Petra Vidnerová, The Czech Academy of Sciences, Institute of Computer Science* and is available on github. The code had an initializer for the centers using Kmeans but no initializer for the dispersion of each neuron. Hence the *InitBetas* class was created where the variance is initialized for each neuron (The code uses the notation beta instead of the variance where beta = $\frac{1}{2\sigma^2}$).

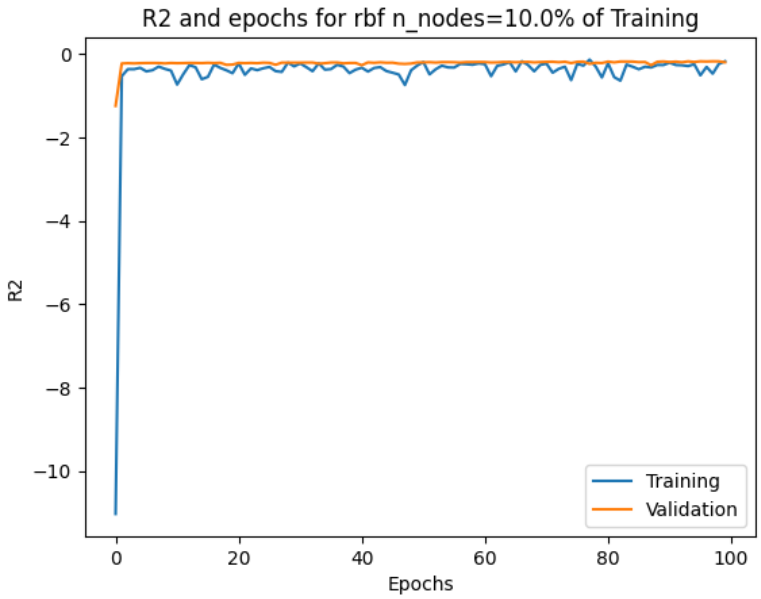
The clusters are calculated with the K-means algorithm, with the maximum distance for each cluster being the one between the centers of the clusters (d_{max}) and the dispersion for each neuron is set to the value $\sigma = \frac{d_{\max}}{\sqrt{2P}}$, where P is the number of output neurons. Moreover, the trainable option for both the centers and the betas is set to False as during training we want only the weights of the output layer to be trained.

Before training the models, the features were standardized based on their mean value and standard deviation. The data was split into 75% training set and 25% testing set.

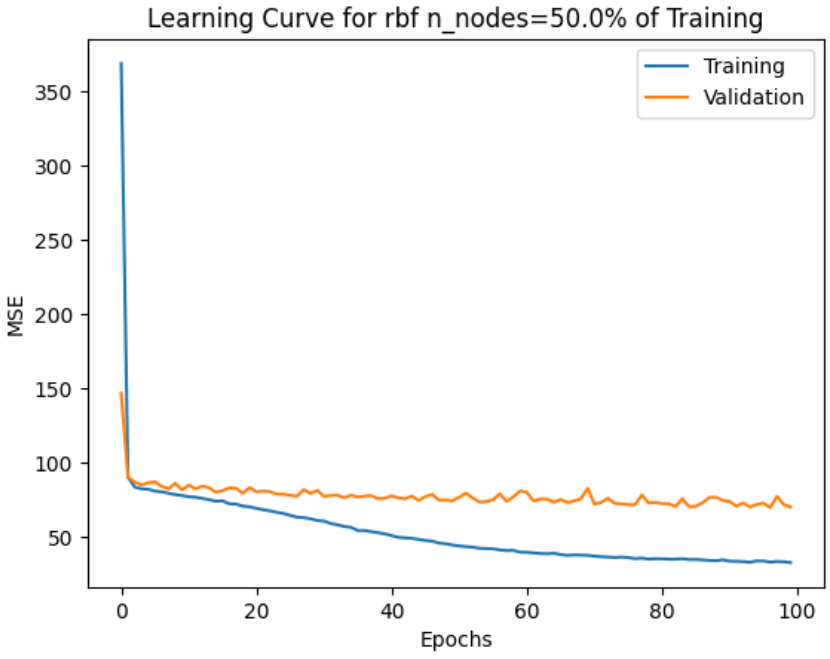
The RBF network comprises three layers: the RBF layer, which applies a Radial Base Function transformation to the input; a layer with 128 nodes for linear transformation of the RBF layer outputs; and a single-node output layer for final network output. SGD was employed as the optimizer, using a learning rate of 0.001, with mean squared error loss, RMSE, and R2 as evaluation metrics. Training occurred over 100 epochs, utilizing a batch size of 6, and 20% of the training set served as the validation set. Three experiments were conducted to determine the optimal number of RBF neurons, corresponding to {10%, 50%, 90%} of the training data population. The learning curves, R2 values for both training and validation data, along with the R2 and RMSE metrics for the testing data, are presented following these experiments.



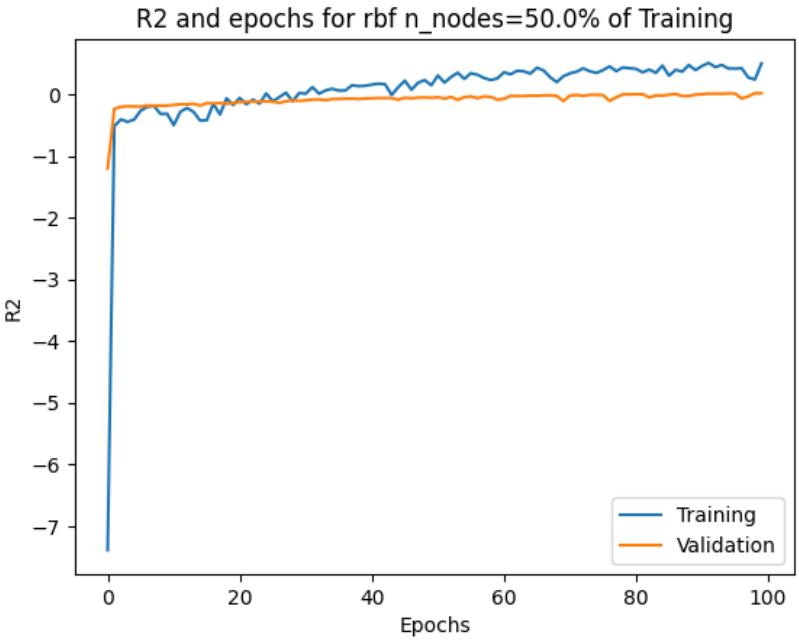
Learning curves for 10% nodes



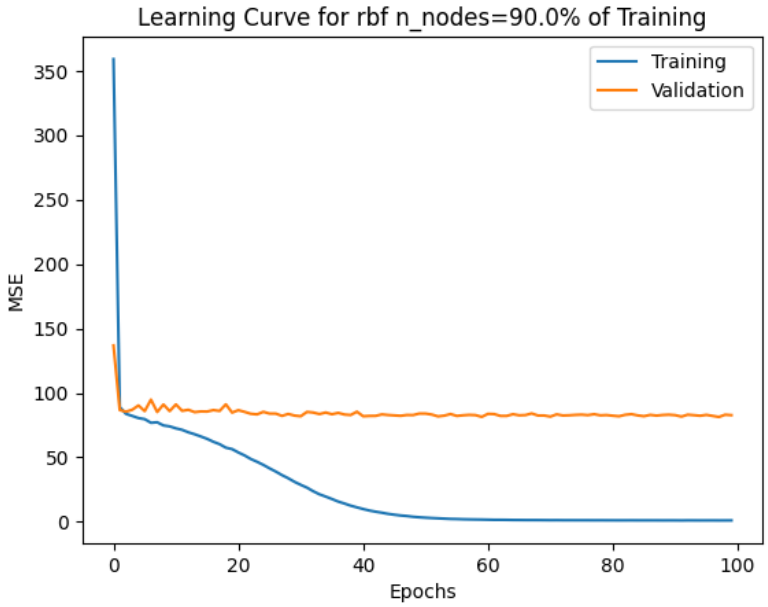
R^2 for training and validation set for training seasons for nodes 10%



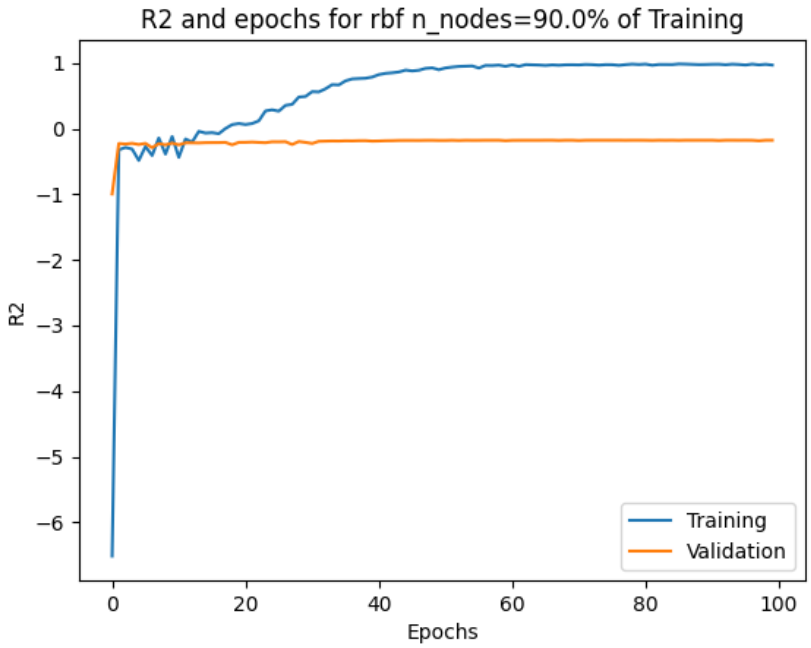
Learning curves for 50% nodes



R^2 for training and validation set for training seasons for nodes 50%



Learning curves for 90% nodes



R ^{2} for training and validation set for training times for nodes 90%

N Nodes/Evaluation Metrics	RMSE	R^2
Nodes 10%	8.2551	-0.1578
Nodes 50%	8.2356	-0.1286
Nodes 90%	7.9687	-0.048

In all three cases, the MSE and R2 values for the validation set remain consistent across different percentages of nodes, except for the 50% case where the validation MSE shows a slight decrease over time. As for the training set MSE, it decreases with an increasing percentage of nodes (80 for 10%, 45 for 50%, and 5 for 90%). Additionally, R2 improves with a higher number of nodes, approaching unity for the 90% case. Consequently, the network with 90% of nodes exhibits the best performance on the training data. Regarding the testing set, metrics show improvement with an increase in the number of neurons, with the 90% network demonstrating the best performance among the three models. However, it's worth noting that the R2 for all three models is less than 0, indicating that the models struggle to make accurate predictions, albeit the 90% nodes model showing the closest approximation to good predictions.

Fine tuning the RBF network

The network chosen for fine-tuning is identical to the previously described one, with the addition of a dropout layer for the neurons in the output layer. SGD was employed as the optimizer with a learning rate of 0.001, utilizing mean squared error loss and RMSE as evaluation metrics. Training occurred over 100 epochs with a batch size of 6, and 20% of the training set was designated for validation purposes. Fine-tuning was conducted using the Keras tuner RandomSearch, with the objective of minimizing validation loss. The table below outlines the hyperparameters considered for tuning and their respective search ranges.

HyperParameters	**Values**
Nodes of RBF layer (percentage)	5%, 15%, 30%, 50% of training set size
Nodes of 2nd ^{hidden} Layer	32, 64, 128, 256
Dropout probability	0.2, 0.35, 0.5

The hyperparameters as well as the MSE for the validation set for the 5 best models with fine tuning are shown in the table below.

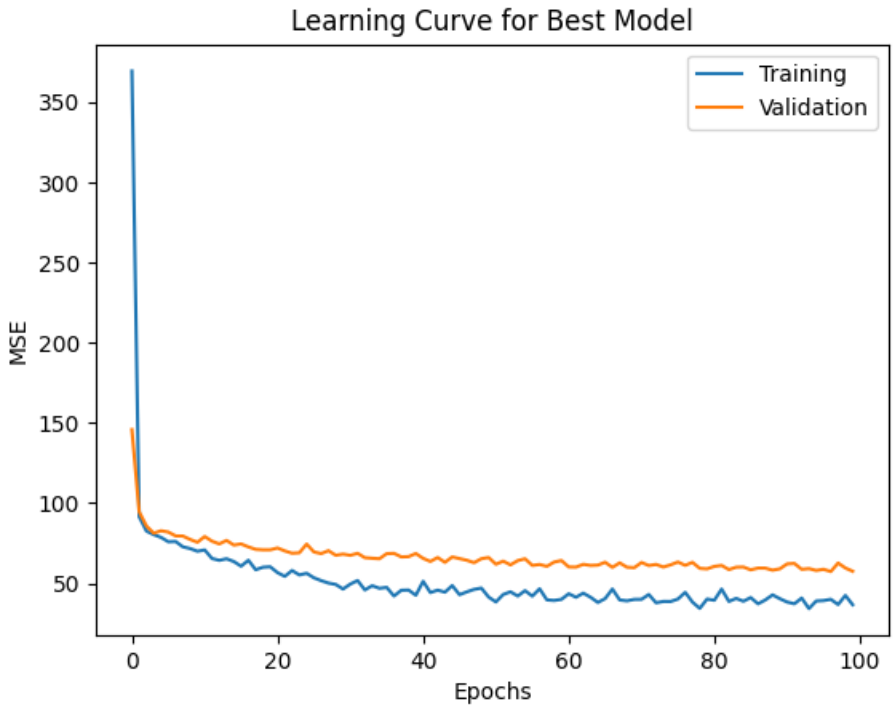
Nodes of RBF Layer (percentage)	**Nodes of ^{2nd} Layer**	**Dropout Rate**	**MSE Validation Set**
50%	**32**	**0.35**	**45.19**
30%	32	0.35	45.51
15%	32	0.35	47.67
50%	64	0.2	48.37
15%	32	0.2	56.04

The top three best models exhibit variations solely in the number of RBF neurons, with performance consistently improving as the number of neurons increases. Furthermore, the optimal performance is achieved with the smallest possible number of output neurons and an

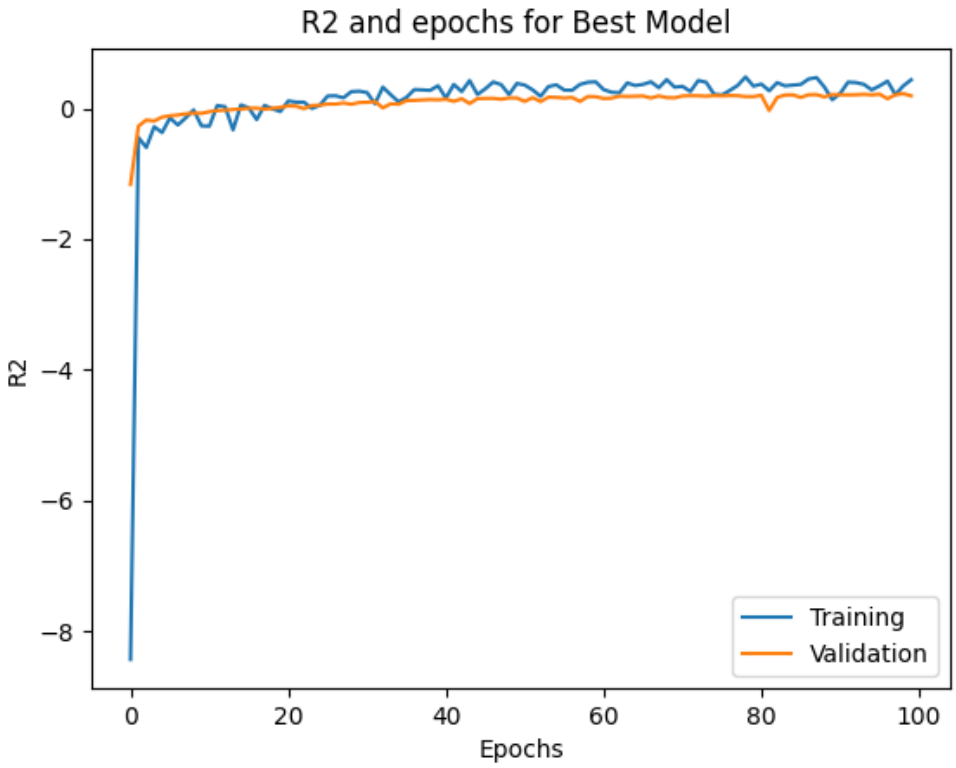
intermediate dropout rate. Consequently, the specifications for the best model are as follows.

HyperParameters	**Values**
Nodes of RBF layer (percentage)	50% of training set size
Nodes of 2nd ^{hidden} Layer	32
Dropout probability	0.35

Below learning curves and R^2 for the training and validation data as well as the R^2 and RMSE metrics for the testing data for the model with the best hyperparameter values are presented.



Learning curves



R^2 for training and validation set for training seasons

Evaluation Metrics	RMSE	R^2
Testing Data	7.271	0.1816

We observe that the MSE for the training set is lower than that of the validation set, while the R2 values are similar for both sets. Additionally, the model performs well on the testing data, showing better performance compared to the models in the first question. Notably, the R2 for the testing data exceeds 0, indicating the model's ability to make effective predictions for unknown data.