Xi'an Jiaotong-Liverpool University
西交利物浦大学

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

Assessment 3

Sahand Sabour

Lecturer

Dr. Rui Yang

CSE301

Bio-computation

Student ID

1614650

Introduction

Machine learning has become an essential task in modern-day Artificial intelligent (AI) systems; it can be defined as the machine's ability to learn a given task on its own rather than being programmed for such operations. There exists a plethora of machine learning algorithms; Due to the scope of the CSE301 module, they would be mainly divided into two types for this assignment: supervised and unsupervised learning. In supervised learning, a mapping between given inputs and outputs would be created in order to map new examples while unsupervised learning allows for classification of unknown maps (unlabeled data). There are benefits to both of these types and their performance varies depending on the type of application.

In this assignment, two artificial neural networks (ANN) based on a back-propagation (BP) trained multi-layer perceptron (MLP) and radial basis function (RBF) structures are created respectively and analyzed via modifications of their corresponding parameters. For the latter structure, k-means clustering algorithm is used to find the RBF centers. These networks are constructed using Matlab and are trained and tested with a given data-set, which includes 117 samples of 5 different car logos, containing 80 features for each sample. As instructed, 80% of the samples are used to train the networks while the remaining 20% are used to test their performance. The created networks illustrate the implementation of both supervised and supervised learning, with MLP being fully supervised and RBF being a combination of the two. Detailed explanation for both of these architectures as well as the experimental results regarding the networks' performance are provided in the following sections.

Methodology

In this section, the methods and models that are implemented in this assignment are described and corresponding training and testing algorithms are provided respectively.

Multi-Layer Perceptron (MLP)

Perceptrons are neural networks that use the error-correcting rule to update their weights in order to learn a mapping. In this architecture, the corresponding output (referred to as the target) for each given input is provided to the network and hence, the network is taught (supervised to learn) a given pattern. However, perceptrons were limited to merely learn patterns that were linearly separable, which is not the case in many real-life applications. Hence, a hierarchical structure of multiple layers of perceptrons, known as MLP, was introduced. By utilizing multiple layers and non-linear activation functions, such as sigmoid and hyperbolic tangent, MLPs were enabled to learn non-linear patterns.

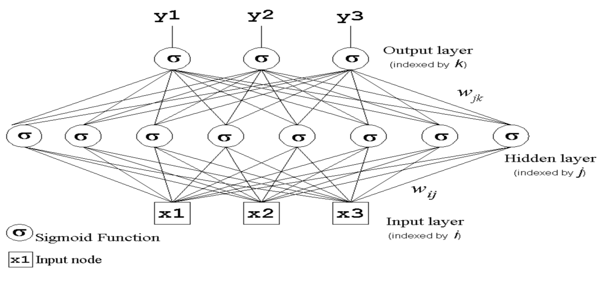


Figure 1: Multi-layer perceptron structure with one hidden layer [1]

Figure 1 illustrates the structure of a MLP network with a single hidden layer, which is similar to the network that was created for this assignment. Error backpropagation learning algorithm, which is of supervised type, is used to train the network and it consists of two passes along the network: 1) Forward pass; 2) Backward pass. In the prior pass, the outputs are calculated using the sigmoid function. Hence, for each sample (x,y), we would have

$$o_j = \sigma(s_j) = \sigma(\sum_{i=0}^{N} w_{ij}x_i) \textbf{ and } o_k = \sigma(s_k) = \sigma(\sum_{i=0}^{N} w_{jk}o_j) \tag{1}$$

where o_j and o_k are the outputs for hidden unit j and output unit k respectively.
The backward pass is made through the network from the output layer to the input and computes the required changes in the weights and updates them accordingly. More specifically, for each output unit k, benefit β_k is calculated.

$$\beta_k = o_k(1 - o_k)(y_k - o_k) \tag{2}$$

where y_k is the desired target for that unit. Consequently, the required changes in the weights between the hidden and output layer are calculated as

$$\Delta w_{jk} = \eta \beta_k o_j \textbf{ and } \Delta w_{0k} = \eta \beta_k \tag{3}$$

with η being the learning rate and Δw_{0k} being the change in the hidden-to-output bias' weight. Accordingly, β_j would be calculated as follows

$$\beta_j = o_j(1 - o_j)(\sum_k \beta_k w_{jk}) \tag{4}$$

Subsequently, the weights changes between the input and hidden layer are calculated as

$$\Delta w_{jk} = \eta \beta_j x_i \textbf{ and } \Delta w_{0k} = \eta \beta_j \tag{5}$$

Conclusively, the weights are updated as $W = W + \Delta W$. In general, ΔW can be derived as $-\frac{\delta E}{\delta w(t)}$, with $w(t)$ being the current weight change. This would result in $o(1 - o)$ if sigmoid activation functions are used. According to (3) and (5), it can be observed that the learning rate plays an essential role in determining the required weight changes. Hence, the goal is to find the largest learning rate that gives the best solution, as smaller learning rates would have slower progress and very large learning rates lead to less accurate results. Consequently, Momentum could be added as an additional parameter for determining ΔW. Since the weight changes tend to occur in a continuous manner, the addition of momentum would create a certain amount of inactivity. This is highly beneficial as it allows the weight changes to be more smooth and less susceptible to errors, and it enables the network to escape from local minima points on the error surface. In addition, if the weight changes occur in the same direction, addition of momentum can lead to faster convergence as it magnifies the learning rate. Therefore, $\Delta w(t)$ for index t of the current weight change with momentum α can be written as

$$\Delta w(t) = -\frac{\delta E}{\delta w(t)} + \alpha \Delta w(t - 1) \tag{6}$$

The summarized training algorithm for MLP is provided in Algorithm1.

Algorithm 1: MLP Training Algorithm

```
Input: Labeled training samples;
Initialization: Set weights to small random values;
while haven't reached max number of epochs and have weight changes do
    Forward pass: calculate outputs at hidden and output nodes;
    Backward pass:;
        1. calculate benefit $b_k$
        2. calculate weight changes $w_{jk}$ and $w_{0k}$
        3. calculate benefit $b_j$
        4. calculate weight changes $w_{ij}$ and $w_{0j}$
        5. update the weights ($W = W + \Delta W$)
end
```

Radial Basis Function Network (RBFN)

The design of neural networks can be considered as a curve-fitting problem. Hence, radial basis functions, whose values merely depends on the distance from the origin and can be modified to depend on the distance from a given center, can be used to achieve this objective. RBF-based networks consist of a single hidden layer in addition to the input and output layers. The structure of such network is displayed in the below figure (Figure 2)

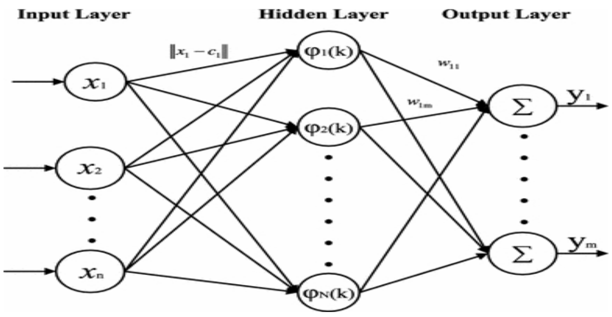


Figure 2: Radial basis function network structure

As illustrate in the figure, non-linear activation functions are used between input and hidden layers while linear activation functions are used between the hidden and output layers. Hence, this structure is able to convert the samples to higher dimensions, where they are more likely to be linearly separable. Consequently, linear mapping could be utilized to transform the obtained results from the hidden layer to the output.

The Gaussian basis function is used in this assignment. As previously mentioned, this function can be modified to depend on given centers rather than the origin. Hence, the Gaussian basis function for $\phi(x)$ can be written as $\phi(x) = e^{-\frac{\|x-c\|^2}{2\sigma^2}}$, where c is a given center and σ^2 is the variance. Accordingly, the interpolation function $F(x)$ for the RBFN would be

$$F(x) = \sum_{i=1}^{n} w_i \phi_i(x) = \sum_{i=1}^{n} w_i e^{-\frac{\|x-c_i\|^2}{2\sigma^2}} \tag{7}$$

where n is the number of units in the hidden layer.

The RBF learning algorithm consists initially determines the basis functions with corresponding centers and variances. In this assignment, k-means clustering algorithm is used to detect the centers based on n number of units in the hidden layer. This method would initially randomly assign the input data-points to one of the n clusters. Consequently, the mean of each cluster is calculated. Subsequently, the distance of each data-point to each center is calculated and in the case that the minimum distance is between the data-point and its current center, it would be assigned to its closest center. This process continues until no further updates occur. As there are no defined targets (correct clusters) for each data-point, this method is considered to be of unsupervised type. After obtaining the RBF centers, the outputs for each example is

$$\phi_{ei}(x) = e^{-\frac{\|x_e-c_i\|^2}{2\sigma^2}} \tag{8}$$

Accordingly, the design matrix Φ for N samples in a RBF with n hidden units would be

$$\Phi = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} \\ & & \cdots & \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{Nn} \end{bmatrix} \tag{9}$$

Since we have that $\Phi W = d$, with W and d being the weights and target vectors respectively, it can be derived that

$$W = (\Phi^T \Phi)^{-1} \Phi^T d = \Phi^+ d \tag{10}$$

where Φ^+ is the pseudo-inverse of Φ. Based on the above equations, it can be observed that the RBFN in this assignment implements both supervised and unsupervised learning techniques. Moreover, it can be noticed that number of RBF centers plays an important role in its performance and therefore, experimental results are to be used to determine its optimal value. The summarized learning algorithm for RBF is provided in Algorithm2.

Algorithm 2: RBF Learning Algorithm

```
Input: Labeled training samples;
Initialization: Define number of RBF centers;
begin
Find RBF centers using k-means;
foreach example e of N examples do
    foreach center i of n RBF centers do
        Calculate the output $\phi_{ei}$;
    end
end
Create design matrix $\Phi$;
Find its pseudo-inverse $\Phi^+$;
Compute weights vector $W$;
end
```

Comparison of MLP and RBFN

Both the MLP and the RBF architectures are feed-forward networks that use non-linear activation functions and are proven to provide significant performance. However, based on the mentioned description of each model, there are a number of differences between the two, which are provided respectively below.

1. RBF network has only one hidden layer whereas MLP can have multiple hidden layers.
2. MLP uses a number of hidden layers to learn non-linearly separable patterns while RBF increases the input dimensions in order to make the pattern linearly separable.
3. The activation functions in the hidden and output layers of MLP are both non-linear functions whereas only the hidden layers' nodes would have non-linear transfer functions.
4. RBF uses different activation functions ($\phi(x)$) at each node while nodes within hidden and output layers in MLP share the same activation function.
5. The Gaussian basis model in RBF calculates the distance between inputs and centers whereas the inner product of input and its corresponding weight is computed in MLP.

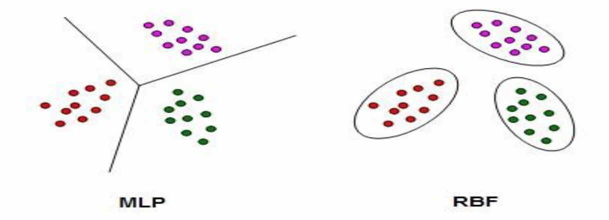


Figure 3: Clustering comparison of MLP and RBF [2]

Hence, MLP would be fitting for global approximations as the target are determined in prior whereas RBF would be fitting for local approximations since the target clusters are unknown and therefore, are determined by the number of centers. Figure 3 illustrates this point, as it shows the different classification results of the mentioned methods.

Experimental Results and Analysis

Initially, the given data-set is divided to 5 classes representing the vehicle logos respectively. Accordingly, 80% of the samples for each class were used to create the training data while the remaining 20% were decided as the data for testing. Consequently, both the MLP and the RBF structures were created using Matlab and trained with the training data samples, and their performance for different parameters were recorded and analyzed. Based on the obtained performances, the best model for each architecture was chosen and the corresponding confusion matrix was delineated. The obtained results and relative discussion is provided below.

Multi-Layer Perceptron (MLP)

The learning rate and momentum of this network were initially set to 0.1. Subsequently, the network's performance with different number of units in the hidden layer was analyzed.

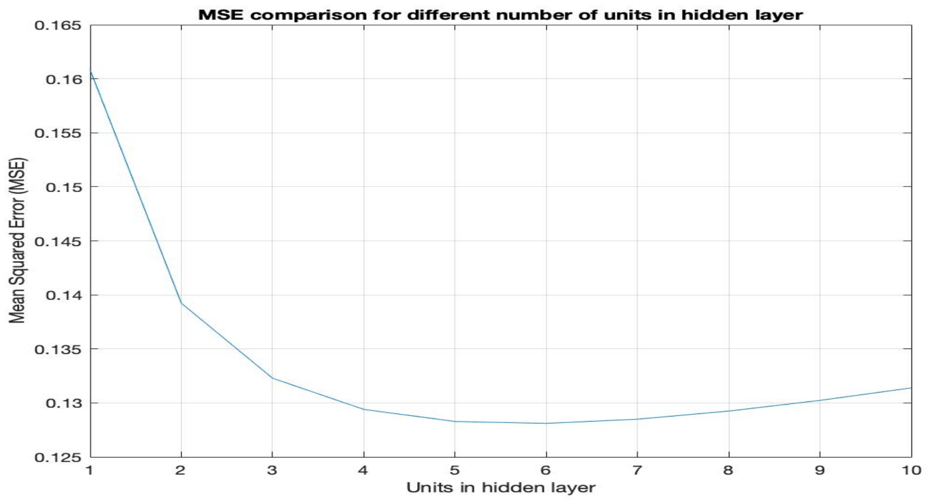


Figure 4: MSE comparsion for different number of hidden units

Based on the above results, it can be observed that the increase in the number of units in the hidden layer initially has a relative correlation with the overall performance of the network: more units would lead to less error; however, this correlation would not be true after a certain number of units. Therefore, more units in the hidden layer would not result in better performance, which was as expected. In addition, the optimal number of hidden units for the best model can be obtained from the figure. Consequently, different values of learning rate for the initial momentum and the optimal number of hidden units were investigated.

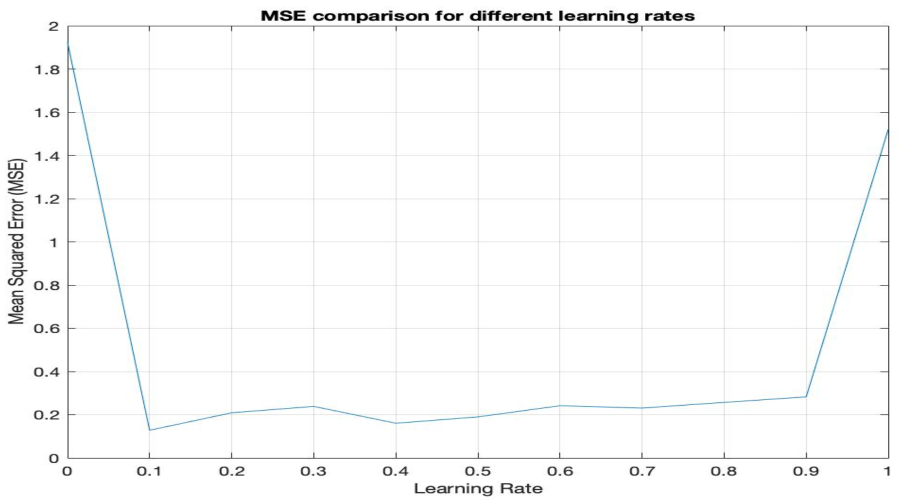


Figure 5: MSE comparsion for different learning rates

Figure 5 displays that different values of the learning rate, excluding 0 and 1, would achieve nearly similar performance. However, when the learning rate it set to 0, the error would be maximized as there is no learning involved in the network. In addition, the network's performance is significantly poor when the learning rate is 1 as it leads to considerably faster convergence. Consequently, by obtaining the optimal values for number of hidden units and learning rate, different values of momentum were to be analyzed to find the best MLP model.

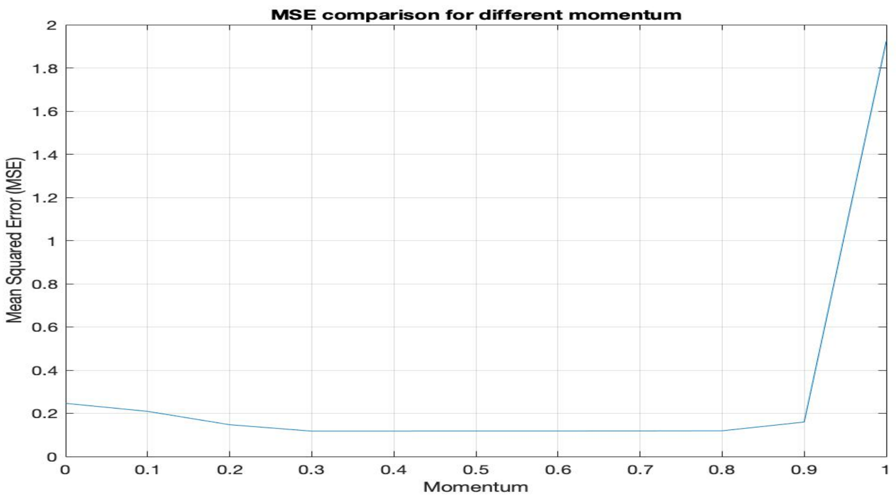


Figure 6: MSE comparsion for different momentum

As displayed in Figure 6, changes in the momentum do not considerably change the network's performance and would merely increase the convergence speed as long as the momentum is not set to 1. In this case, based on (6), the gradient descent would be ignored and weight change would be solely dependent on the value of momentum, which is clearly shown by the above figure. Conclusively, by the above simulations, the best MLP model could be obtained.

Radial Basis Function Network (RBFN)

The network's performance based on different number of RBF centers was analyzed and the obtained results are provided in the below figure (Figure 7).

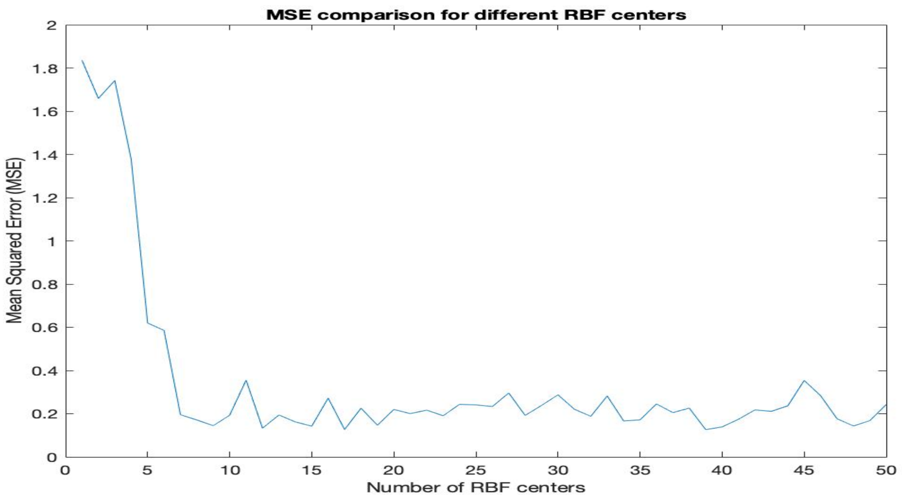
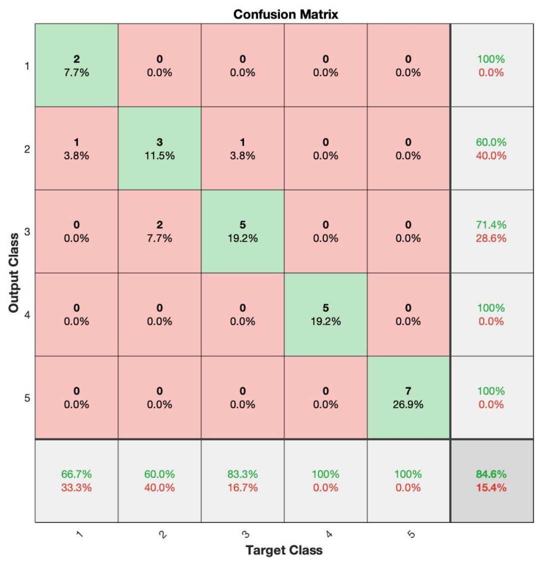


Figure 7: MSE comparsion for different number of RBF centers

The above figure illustrates that higher performance can be achieved when the number of centers are more than the actual target: In this case, there are 5 classes and the error is considerably decreased with 6 or more centers. As theorized, the fitting value would be close to the mean of number of inputs and number of outputs. Conclusively, the best RBF model would be obtained.

Comparison of MLP and RBFN

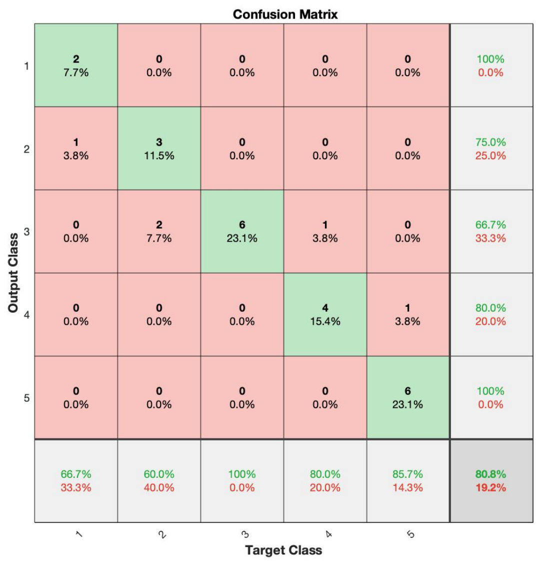
The confusion matrix for best MLP and RBF models are provided in Figure (8).



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	
1	2 7.7%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
2	1 3.8%	3 11.5%	1 3.8%	0 0.0%	0 0.0%	60.0% 40.0%
3	0 0.0%	2 7.7%	5 19.2%	0 0.0%	0 0.0%	71.4% 28.6%
4	0 0.0%	0 0.0%	0 0.0%	5 19.2%	0 0.0%	100% 0.0%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	7 26.9%	100% 0.0%
	66.7% 33.3%	60.0% 40.0%	83.3% 16.7%	100% 0.0%	100% 0.0%	**84.6%** **15.4%**

(a) Best MLP model



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	
1	2 7.7%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
2	1 3.8%	3 11.5%	0 0.0%	0 0.0%	0 0.0%	75.0% 25.0%
3	0 0.0%	2 7.7%	6 23.1%	1 3.8%	0 0.0%	66.7% 33.3%
4	0 0.0%	0 0.0%	0 0.0%	4 15.4%	1 3.8%	80.0% 20.0%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	6 23.1%	100% 0.0%
	66.7% 33.3%	60.0% 40.0%	100% 0.0%	80.0% 20.0%	85.7% 14.3%	**80.8%** **19.2%**

(b) Best RBF model

Figure 8: Confusion matrix for best models

The above results display that the optimal performance of the networks were considerably similar. Although it can be observed from the figure that the MLP network achieves higher accuracy, it can not be claimed that this would be the general case. This is due to the fact that the RBF centers are randomly initialized and therefore, they can highly affected the k-means results. Hence, the only statement to be made is that in this assignment, given the implemented algorithms and Matlab code, the best MLP network was able to outperform the best RBF network.

Conclusion

In this assignment, respective multi-layer perceptron (MLP) and radial basis function (RBF) based networks were created. Thorough description of each model as well as corresponding training algorithms were provided. The models were trained and tested using the given data-set and their performance with different parameters were analyzed and recorded. Consequently, the best MLP and RBF model were decided and their performance were compared. It is believed that based on the theories that were instructed in the CSE301 modules, the experiment was successfully conducted and the results are reliable. This assignment is considered as a valuable opportunity to delve into the course and gain better understanding of the teaching material.

References

[1] R. Yang, 2019, Lecture 5: MULTI-LAYER PERCEPTRON, lecture notes, CSE301: Bio-computation, Xi'an Jiaotong Liverpool University, delivered 23 Sep. 2019.

[2] R. Chandradevan, (2017, Aug. 8) *Radial Basis Functions Neural Networks — All we need to know*, [Online]. Available: https://towardsdatascience.com/radial-basis-functions-neural-networks-all-we-need-to-know-9a88cc053448