Radial basis function neural networks

In the realm of machine learning and artificial intelligence, Neural Networks (NN) have established their prominence due to their remarkable ability to learn from data and make predictions or decisions without being explicitly programmed to perform the task. Among various types of neural networks, **radial basis function neural networks (RBFNN)** are a unique class that have proved to be highly effective in various applications including function approximation, time series prediction, classification, and control.

In this Answer, we will comprehensively explore the constituents and functionality of radial basis function neural networks.

What are radial basis function neural networks?

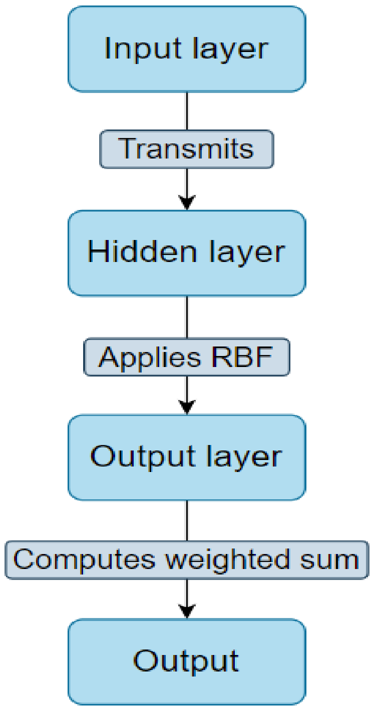
A radial basis function (RBF) neural network is a type of artificial neural network that uses radial basis functions as activation functions. It typically consists of three layers: an input layer, a hidden layer, and an output layer. The hidden layer applies a radial basis function, usually a Gaussian function, to the input. The output layer then linearly combines these outputs to generate the final output. RBF neural networks are highly versatile and are extensively used in pattern classification tasks, function approximation, and a variety of machine learning applications. They are especially known for their ability to handle non-linear problems effectively.

Structure of RBF neural networks

An RBF neural network typically comprises three layers:

- **Input layer**: This layer simply transmits the inputs to the neurons in the hidden layer.
- **Hidden layer**: Each neuron in this layer applies a radial basis function to the inputs it receives.
- **Output layer**: Each neuron in this layer computes a weighted sum of the outputs from the hidden layer, resulting in the final output.

Here's the basic flow diagram of the RBF neural network:



Structure of RBF neural networks

Mathematical background

The output y of an RBF network is a linear combination of radial basis functions. It is given by:

$$y(x) = \sum_{i=1}^{N} w_i \phi(\|x - c_i\|)$$

where:

- *x* is the input vector
- *N* is the number of neurons in the hidden layer
- *wi* are the weights of the connections from the hidden layer to the output layer
- *ci* are the centers of the radial basis functions

- $\|x - c_i\|$ is the Euclidean distance between the input vector and the center of the radial basis function
- ϕ is the radial basis function, usually chosen to be a Gaussian function:

$$\phi(r) = e^{-\beta r^2}$$

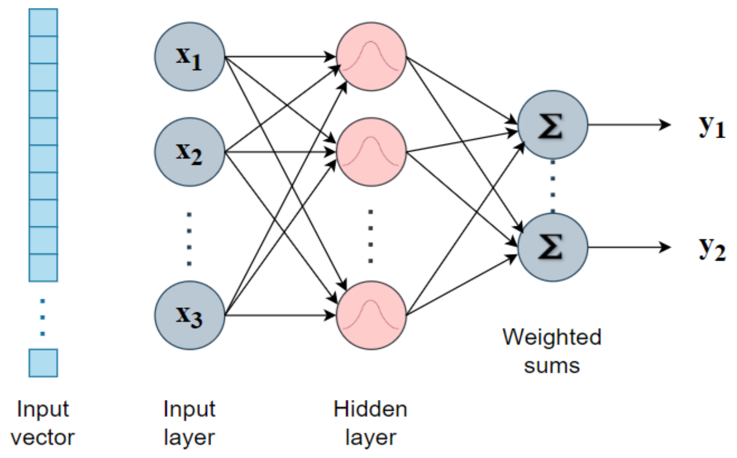
How do RBF neural networks work?

Radial basis function networks (RBFNs) work by comparing the input to known examples from the training data to classify it.

Here’s a simplified explanation:

1. RBFNs start with an input vector. This vector is fed into the input layer of the network.
2. The network also has a hidden layer, which comprises radial basis function (RBF) neurons.
3. Each of these RBF neurons has a center, and they measure how close the input is to their center. They do this using a special function called a Gaussian transfer function. The output of this function is higher when the input is close to the neuron’s center and lower when the input is far away.
4. The outputs from the hidden layer are then combined in the output layer. Each node in the output layer corresponds to a different category or class of data. The network determines the input’s class by calculating a weighted sum of the outputs from the hidden layer.
5. The final output of the network is a combination of these weighted sums, which is used to classify the input.

Here's a visual representation of the above explanation:



Working of RBF neural network

Training an RBF network

Training an RBF network involves two steps:

1. **Determining the centers c_i and the parameter β of the radial basis functions**: This can be done using a clustering algorithm like K-means on the training data.
2. **Determining the weights w_i**: This can be done using a linear regression algorithm on the outputs of the hidden layer.

Python implementation

Here's a simple implementation of a Radial Basis Function Network (RBFN) using Python. This code creates a simple RBFN and trains it on some dummy data.

This implementation will use:

- KMeans for clustering and defining the radial basis functions.
- Linear regression for learning the weights.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_classification
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
import scipy.spatial.distance as distance

class RadialBasisFunctionNeuralNetwork:
    def __init__(self, num_of_rbf_units=10):
        self.num_of_rbf_units = num_of_rbf_units

    def _rbf_unit(self, rbf_center, point_in_dataset):
        return np.exp(-self.beta * distance.cdist([point_in_dataset], [rbf_center],
'euclidean')**2).flatten()[0]

    def _construct_interpolation_matrix(self, input_dataset):
        interpolation_matrix = np.zeros((len(input_dataset), self.num_of_rbf_units))
        for idx, point_in_dataset in enumerate(input_dataset):
            for center_idx, rbf_center in enumerate(self.rbf_centers):
                interpolation_matrix[idx, center_idx] = self._rbf_unit(rbf_center, point_in_dataset)
        return interpolation_matrix

    def train_model(self, input_dataset, target_dataset):
        self.kmeans_clustering = KMeans(n_clusters=self.num_of_rbf_units,
random_state=0).fit(input_dataset)
        self.rbf_centers = self.kmeans_clustering.cluster_centers_
        self.beta = 1.0 / (2.0 * (self.kmeans_clustering.inertia_ / input_dataset.shape[0]))
        interpolation_matrix = self._construct_interpolation_matrix(input_dataset)
        self.model_weights =
np.linalg.pinv(interpolation_matrix.T.dot(interpolation_matrix)).dot(interpolation_matrix.T).dot(
target_dataset)

    def predict(self, input_dataset):
        interpolation_matrix = self._construct_interpolation_matrix(input_dataset)
        predicted_values = interpolation_matrix.dot(self.model_weights)
        return predicted_values


if __name__ == "__main__":
    # Generating a simple classification dataset
    input_dataset, target_dataset = make_classification(n_samples=500, n_features=2,
n_informative=2, n_redundant=0, n_classes=2)

    # Initializing and training the RBF neural network
    rbf_neural_network = RadialBasisFunctionNeuralNetwork(num_of_rbf_units=20)
    rbf_neural_network.train_model(input_dataset, target_dataset)
    # Predicting the target values
    predictions = rbf_neural_network.predict(input_dataset)

    # Converting continuous output to binary labels
    binary_predictions = np.where(predictions > 0.5, 1, 0)

    # print("Accuracy: {}".format(accuracy_score(target_dataset, binary_predictions)))
    print(f"Accuracy: {accuracy_score(target_dataset, binary_predictions)}")

    # Plotting the results
    plt.scatter(input_dataset[:, 0], input_dataset[:, 1], c=binary_predictions, cmap='viridis',
alpha=0.7)
    plt.scatter(rbf_neural_network.rbf_centers[:, 0], rbf_neural_network.rbf_centers[:, 1], c='red')
    plt.title('Classification Result')
    plt.show()
```

Code explanation

Let’s break down the code:

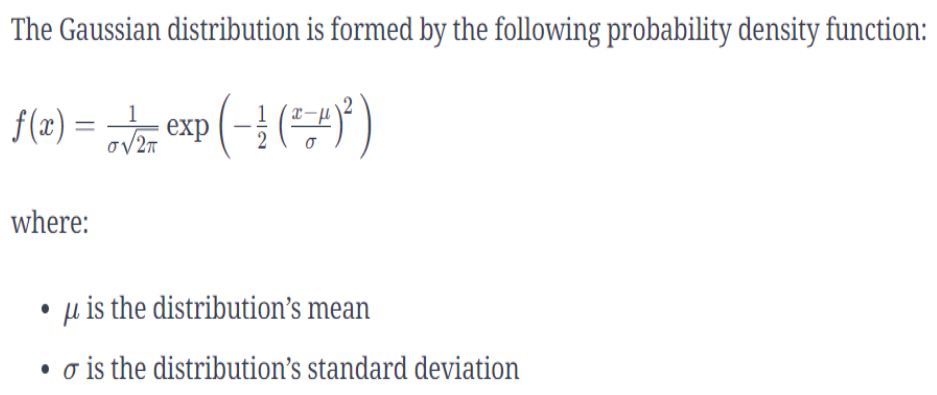
- **Lines 1–7**: Importing the required libraries. We use numpy for numerical operations, matplotlib for plotting, sklearn.cluster.KMeans for unsupervised clustering, sklearn.datasets.make_classification for creating a classification dataset, sklearn.linear_model.LinearRegression for the linear regression model, sklearn.metrics.accuracy_score for evaluating the model, and scipy.spatial.distance for calculating distances.
- **Lines 9–11**: Defining the RadialBasisFunctionNeuralNetwork class and its constructor. The class represents a radial basis function neural network, and its constructor takes one argument - the number of radial basis functions (hidden units).
- **Lines 13–14**: The _rbf_unit method is defined. This is a helper function to calculate the output of a radial basis function (RBF). It computes the Euclidean distance between a point and an RBF center, squares it, multiplies it by a negative beta, and finally applies the exponential function.
- **Lines 16–21**: The _construct_interpolation_matrix method is defined. It creates an “interpolation matrix” where each entry corresponds to the output of an RBF given an input data point. This matrix is needed to compute the weights in the RBFNN.
- **Lines 23–28**: The train_model method is defined. It first uses KMeans clustering to find the centers of the RBFs. It then computes beta based on the average squared distance between data points and their nearest cluster center (the inertia). Finally, it computes the weights that connect the RBFs to the output layer using a pseudoinverse of the interpolation matrix and the target values.
- **Lines 30–33**: The predict method is defined. It computes the interpolation matrix for the input data and uses it with the weights to predict the output values.

- **Lines 36–46**: The main program execution starts. A classification dataset is generated using the make_classification function from sklearn. Then an instance of the RBFNN is created and trained on the data. The trained model is then used to predict labels for the input data.
- **Line 48**: The continuous output from the predict function is converted to binary class labels based on a threshold of 0.5.
- **Line 51**: The accuracy of the model on the training data is printed to the console.
- **Lines 54–57**: A scatter plot of the data is created, where the color of each point indicates its predicted class label. The centers of the RBFs are also plotted as red points. The plot is displayed using plt.show().

Radial basis function neural networks are powerful tools for function approximation problems. They are relatively simple to implement and can model complex non-linear relationships. However, they require careful tuning of their parameters and may not be suitable for high-dimensional data due to the curse of dimensionality.

Gaussian distribution, commonly referred to as the normal distribution, is a commonly used continuous probability distribution. Many types of data can be fitted using Gaussian distribution, such as the heights of a population and the test scores of students.

The Gaussian distribution is formed by the following probability density function:



The Gaussian distribution is formed by the following probability density function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

where:

- μ is the distribution's mean
- σ is the distribution's standard deviation