Introduction to Working with Artificial Neural Networks

- **Author:** Arshia Goshtasbi
- **GitHub:** @Arshiagosh

Introduction

Consider a number of functions (at least 3 functions and a maximum of infinite) with one-dimensional input from simple to complex. For example, consider a specific sine function and generate some points on this function in a specific domain. Consider these points as the training set and the actual values of these points in the original function as the test set, and try to extract the function's output using a neural network (MLP).

Tasks

1. Plot the function learned by the network alongside the correct function using the Plot library and calculate and evaluate the error of what has been learned (with any criteria).
2. To plot the function learned by the network, provide small and close points in the input domain (preferably broader than the training data domain) as inputs to the network and obtain and display the output.
3. Experiment with the following parameters and analyze their impact on the results:

- Number of input points
- Complexity of the desired function
- Number of network layers and number of neurons in each layer
- Number of network cycles to complete learning
- Input domain extent, especially in more complex functions
- Any other parameter you think is effective

4. The function should be multivariate.
5. Display the real function and the predicted function on a plot.
6. Calculate MSE (Mean Squared Error) and MAE (Mean Absolute Error).
7. Use Keras library for implementing the neural network.

Report

The provided code demonstrates the implementation process of approximating various functions using a multilayer perceptron (MLP) neural network. The code is divided into four main sections, each focusing on a different function:

1. Simple Function: `f(x) = x^2`
2. Polynomial Function: `f(x) = x^3 - 2x^2 + 3x - 1`
3. Trigonometric Function: `f(x) = sin(2x) + cos(3x)`
4. Complex Function: A piecewise function with different expressions for different intervals of the input domain.

For each function, the code follows a similar process:

1. Generating synthetic training and test data for the function.
2. Scaling the input data using MinMaxScaler.
3. Defining the neural network architecture with Keras.
4. Compiling the model with mean squared error loss and Adam optimizer.
5. Training the model with cross-validation (5 folds) and tracking the best fold based on validation loss.
6. Plotting the true function and the predicted function for the best fold.
7. Plotting the training and validation loss for all folds.

The code calculates and reports the Mean Squared Error (MSE) and Mean Absolute Error (MAE) for each fold and the average across all folds. These metrics are used to evaluate the accuracy of the learned function.

Experiments and analyses can be conducted by modifying various parameters, such as:

1. **Number of input points:** Increasing the number of input points generally leads to better approximation, but may also increase the computational cost and risk of overfitting.

2. **Complexity of the desired function:** More complex functions may require deeper or wider neural networks to capture the intricate patterns.

3. **Number of network layers and number of neurons in each layer:** Deeper networks with more neurons can better approximate complex functions, but may also be more prone to overfitting and require more training data.

4. **Number of network cycles to complete learning:** Increasing the number of training epochs can improve the accuracy, but there is a risk of overfitting if training is continued for too long.

5. **Input domain extent, especially in more complex functions:** Functions with discontinuities or rapidly changing behaviors may require a broader input domain to ensure the neural network can learn the patterns accurately.

The impact of these parameters on the accuracy and training conditions can be observed and analyzed.

Comparison with Other Methods

The importance of this project lies in analyzing and comparing the neural network method in terms of accuracy and training conditions with other methods that will be covered in the subsequent semester and remaining projects.

In the context of function approximation, the neural network approach can be compared with other methods such as polynomial regression, spline interpolation, or kernel-based methods. The performance of these methods can be evaluated based on their ability to accurately approximate the function, the complexity of the learned model, and the computational requirements for training and inference.

Additionally, the neural network method can be compared with other machine learning techniques, such as decision trees or support vector machines, in terms of their ability to learn and generalize from data, as well as their interpretability and ease of use.

By analyzing and comparing the results obtained from different methods, we can gain insights into their strengths and weaknesses, and potentially develop hybrid or ensemble approaches that leverage the advantages of multiple techniques.