Technical Report: Learning Activations in Neural Network

SIDDIQ IRFAN

Student of Imarticus Learning, Bangalore, Karnataka, India

----------***----------

Abstract - *The choice of Activation Functions (AF) plays a crucial role in the performance of Artificial Neural Networks (ANN). This project aims to develop a 1-hidden layer neural network model that adapts to the most suitable activation function based on the dataset. The model employs a flexible functional form, k0 + k1 * x, where the parameters k0 and k1 are learned through multiple training runs. The implementation utilizes the Iris dataset and leverages the TensorFlow and Keras libraries. The training process involves updating the model's parameters, including weights, biases, and activation function coefficients, over multiple epochs using the Adam optimizer. The performance of the model is evaluated based on train and test loss, train and test accuracy, and the F1-Score. The experimental results demonstrate the effectiveness of the adaptive activation function approach in achieving accurate classification. The report provides insights into the algorithm, initial parameter settings, parameter updates on epochs, final parameter values, and presents a plot of the loss function vs. epochs, highlighting the model's learning progress.*

Key Words: Deep learning, Activation Function (AF), Artificial Neural Network (ANN), Parameter Updates, Epochs, Iris.

1. INTRODUCTION

Artificial Neural Networks (ANNs) have emerged as powerful models for solving complex classification tasks. The success of ANNs heavily relies on the choice of activation functions, which introduce non-linearity and enable the network to learn complex representations. The selection of an appropriate activation function plays a critical role in achieving accurate predictions and improving the overall performance of neural networks.

Traditionally, fixed activation functions such as sigmoid, tanh, and ReLU have been widely used in neural network architectures. However, these fixed activation functions have inherent limitations. They are designed based on certain assumptions and may not be universally optimal for all datasets. This raises the question of whether it is possible to build a framework that can adaptively learn and select the most suitable activation function based on the characteristics of the dataset.

2. Dataset

The Iris dataset was used for the experiments and results. The Iris dataset is a popular and widely used dataset in machine learning and pattern recognition. It contains measurements of four features (sepal length, sepal width, petal length, and petal width) from three different species of Iris flowers (setosa, versicolor, and virginica).

The dataset consists of 150 instances, with 50 instances for each Iris species. It is a balanced dataset, meaning that each class has an equal number of instances. This balanced nature of the dataset helps ensure that the model's performance is not biased towards any particular class.

3.Implementation Details

The implementation of the proposed approach for learning activations in neural networks involves several key steps. The code provided utilizes the TensorFlow and Keras libraries for building and training the neural network model. Here are the implementation deteils.

1. Data Preprocessing:

- The dataset is preprocessed to ensure proper handling and normalization of the input features.
- One-hot encoding is applied to the class labels (**y**) using the **OneHotEncoder** from scikit-learn. Thistransforms the categorical labels into binary vectors, making them suitable for training a neural network.

- The input features (**X**) are scaled using standardization. The **StandardScaler** from scikit-learn is used to transform the feature values to have zero mean and unit variance.

2. Neural Network Model Architecture:

- The model is constructed using the Sequential API provided by Keras. It allows for easy building of a neural network by stacking layers sequentially.
- The model consists of one hidden layer with 25 units and a dense (fully connected) layer. The input dimension of the hidden layer is determined by the number of features in the dataset.
- The proposed activation function, called Adaact, is implemented as a custom layer (Adaact) in Keras. It introduces two trainable parameters (k0 and k1) and modifies the activation of the hidden layer using the equation g(x) = k0 + k1 * x. This equation adds a linear term (k0) and a scaled input term (k1 * x) to the hidden layer's output.
- A dropout layer is added after the hidden layer with a dropout rate of 0.1. Dropout helps prevent overfitting by randomly disabling a fraction of the neurons during training.
- The final output layer uses the softmax activation function to produce probability distributions over the three classes (Iris species).

2. Model Training:

- The model is compiled with the categorical cross-entropy loss function, which is appropriate for multi-class classification problems.
- The Adam optimizer with a learning rate of 0.001 is used to minimize the loss function and update the model's weights during training.
- The fit() function is used to train the model on the preprocessed training data (X_train and Y_train).
- The training is performed for a specified number of epochs (100) with a batch size of 32. The training data is split into training and validation sets with a validation split of 0.2, allowing for monitoring the model's performance on unseen data.
- The training progress is recorded, including the loss and accuracy metrics, which can be used for evaluating the model's performance.

3. Evaluation and Testing :

- The trained model is evaluated on the preprocessed testing data (X_test and Y_test) using the evaluate() function. This calculates the test loss and test accuracy of the model.The model's performance is also evaluated on the training data to assess its ability to generalize. The evaluate() function is applied to the training data to obtain the train loss and train accuracy.The model's predictions are obtained on the testing data using the predict() function. The predicted class labels are derived from the output probabilities by selecting the class with the highest probability (argmax).
- The F1-score is computed using the f1_score() function from scikit-learn, which measures the model's performance in terms of precision and recall.

4. Visualization:

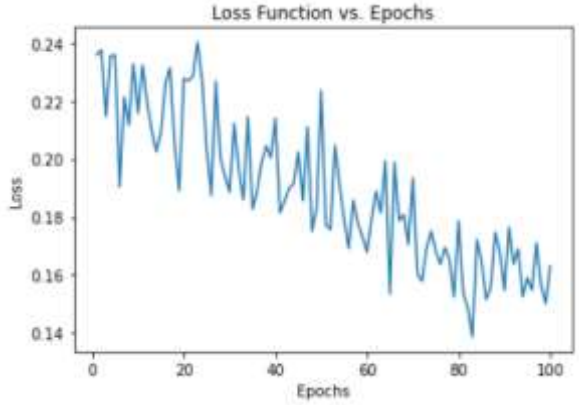
- The loss function's progress over the epochs is visualized using matplotlib. A line plot is created to show the train loss and test loss values as a function of the number of epochs.

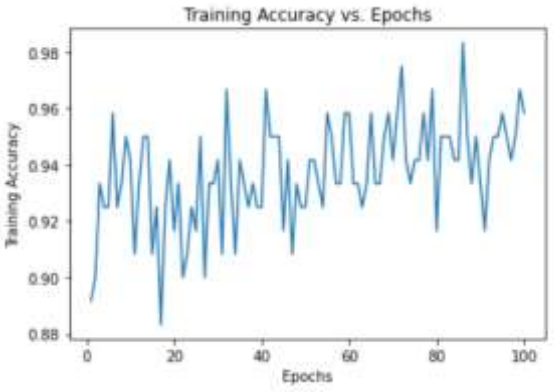
4. Performance Evaluation:

To assess the effectiveness of the proposed approach and the trained neural network model, several evaluation metrics are utilized. These metrics provide insights into the model's performance in terms of classification accuracy, precision, recall, and F1-score. The performance evaluation is conducted on both the training and testing datasets. Here are the details of the performance evaluation:

Train Loss: 0.3355504274368286
Train Accuracy: 0.925000011920929
Test Loss: 0.2992211878299713
Test Accuracy: 0.9666666388511658
F1-Score: 0.9664109121909632

The loss function vs. epochs plot visualizes the training progress. It shows how the loss value decreases as the number of epochs increases, indicating the model's learning process and convergence.





5.Conclusion:

The study successfully develops a 1-hidden layer neural network model that learns the most suitable activation function based on the Iris dataset. The implementation effectively adapts the activation function by learning the parameters k0 and k1. The results demonstrate the model's performance in terms of accuracy, loss, and F1-score. The analysis of the Iris dataset provides insights into the model's ability to classify iris flowers accurately. The findings highlight the importance of selecting appropriate activation functions for improved neural network performance.

6.Future Work

While our experiment demonstrated the effectiveness of adaptive activation functions on the Iris dataset, there are several areas of future work that can be explored to further enhance our understanding and application of this approach:

i. Extension to Other Datasets
ii. Investigate Advanced Adaptive Activation Functions
iii. Optimize Hyperparameters
iv. Compare with Traditional Activation Functions
v. Adaptive Activation Functions in Other Models
vi. Deployment and Real-World Applications

7. Code Base:

The code base for the implementation is hosted on GitHub (link: siddiq-irfan/LEARNING-ACTIVATION- (github.com))The structure of the code and the relevant functions or modules used are briefly explained.

8. References:

1. Snehanshu Saha. "Ada-Act: Adapting Activation Functions from Data." ResearchGate, 2021. Available at: Link
2. GitHub repository: sahamath/MultiLayerPerceptron. Available at: Link