Analysis of Multi-Activation Layers in Neural Network Architectures

Jaskirat Sohal, Braden Stonehill

jsohal@sutdents.kennesaw.edu, bstoneh1@students.kennesaw.edu

https://github.com/bkstonehill/Multi-Activation-Neural-Network

Department of Computer Science, Kennesaw State University

Marietta, GA

Abstract

Neural Network architectures have been commonly constructed as layers that linearly combine inputs and apply an activation function uniformly to all outputs. We introduce a new layer architecture referred to as Multi-Activation Layer, which uniformly distributes activation functions to each node in the layer. This paper explores the effects on performance and runtime costs for applying different activation functions to each node in a layer. The proposed architecture was tested in several problem types and achieved results comparable with identical architectures that applied a single activation function to all outputs from a layer but generalized data in fewer epochs at approximately 50% longer runtime.

Introduction

Neural Networks were created to mimic the functions of a biological brain through a network of computational units that represent the neurons in the brain. Each unit linearly combines input features and applies an activation function to produce non-linear mappings (Bishop, C. for further details). These networks learn patterns and functions through layers of units connected to each other in which the weights for the equations are adjusted by backpropagation of the loss function with gradient descent.

For optimization, the multiple computations of linear combinations and application of activation functions for layers of units are implemented through a series of matrix multiplications and uniform element-wise applications of activation functions on all outputs.

To optimize the computational performance, the multiple executions of linear combinations and activation functions are implemented through a series of dot products. As new architectures for Neural Networks have been designed, they still follow the same principle of transforming input features through multiplication and activation functions, such as the Transformer architecture, which applies multiple linear combinations followed by an activation function for applying attention (Vaswani, A., et al.).

There has been much research into different activation functions for applying non-linear transformations. Different activation functions map values into different ranges for various applications and affect how networks learn. Activation functions allow a network to learn complex patterns and approximate functions, and their derivatives determine how aggressively a network learns. For many network architectures, it is common to use the same activation function throughout every layer, primarily chosen by the task or learning capabilities of the function. However, there has been a study on using different combinations of functions throughout a network (Vijayakumar, K., et al.).

As networks are function approximators, they can be used as an activation function, such as in Modular Neural Networks, where the computational units are also individual Neural Networks to nonlinearly transform features (Ronco, Eric and Gawthrop, Peter, for further details on Modular networks). Modular networks follow a different approach as each unit can be processed through a separate network resulting in different transformations of the features. Using multiple networks can lead to faster computations due to parallel processing. With Modular Neural Networks as an inspiration alongside the study of using different combinations of activation functions, we propose a scalable layer architecture, referred to as a Multi-Activation Layer, which applies varying activation functions to each unit in a network layer.

Related Work

As previously mentioned, there has been a study on applying different activation functions to individual layers of a neural network (Vijayakumar K. et al.). In which the Feed Forward Neural Network was created, implementing varying activation functions for each layer, and compared the results to Uniform Feed Forward Neural Network, which used a single activation function throughout the model. The results indicated improved performance via the application of a combination of functions.

Multiple studies have been conducted on the application of Modular Neural Networks. One of those studies analyzed the network for multiclass classification by using the module to subdivide the problem into networks of two-class problems for each pair of classes (Anand R. et al.). Their experiments found that modular networks converge faster and generalize better than typical networks.

The proposed architecture has properties similar to Modular Neural Networks but has some key differences. The proposed architecture can utilize any activation function, including other networks, but also distributes the activation functions uniformly among all the nodes in a layer. The layer behaves as a typical layer in which the primary hyperparameter for a layer is the number of neurons. Any number of activation functions can be given to the layer, and if a single activation function is specified, it reduces to a simple feed-forward layer. The proposed architecture can also work with activation functions that produce different ranges of output values as the outputs are batch normalized.

Methodology

Activation Functions

Several activation functions were utilized in our model to test the proposed layer architecture. We focused on utilizing commonly used functions to experiment with learning different features from the training data: Sigmoid, Tanh, Leaky ReLU, ELU, and Swish.

The Sigmoid activation function is a commonly used function that maps values to a range of (0, 1) and has applications as a threshold function or for representing probabilities. However, its gradient contributes to the vanishing gradient issue as mappings of large values reduce the gradient to near zero (Ding, B. et al.).

Tanh is an improvement over the sigmoid function. The function maps values from (-1,1), averaging values closer to zero reducing the effects of vanishing gradients but still suffering the same issue as sigmoid with large values resulting in zero gradients.

Leaky ReLU is an improvement over the ReLU activation function. The ReLU activation is an identity function but maps all negative values to zero. This results in zero gradients for negative values and effectively disabling units in a layer (Sharma, S., et al.). Leaky ReLU addresses the issue by scaling negative outputs instead, so the gradient is never zero, preventing units from being disabled when processing values.

ELU is another improvement of ReLU, like Leaky ReLU but smooths negative values nonlinearly. Swish is another improvement over ReLU, which is not monotonic near zero (Sharma, S., et al.) and provides similar improvements as ELU.

The SoftMax activation function scales outputs to probabilities for each class for classification problems.

Loss Functions

To analyze the model's performance in testing, the Cross-Entropy loss function is used for classification problems, and the Root Mean Squared Error loss function is used for regression problems.

Layer Diagram

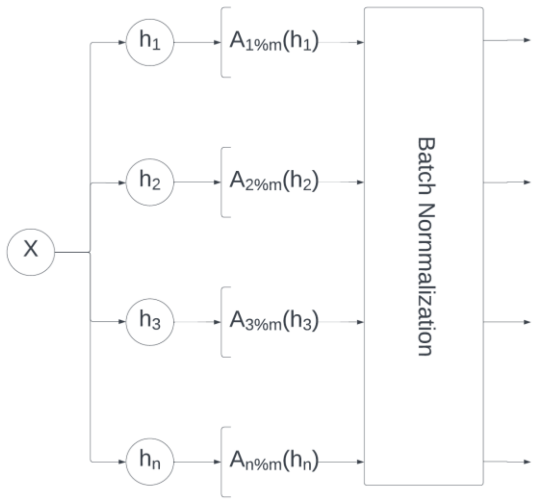

Figure 1 - Diagram of a Multi-Activation Layer

The Multi-Activation layer is the proposed layer in which multiple activation functions are applied uniformly to all computing units in a layer. Figure 1 shows the layer layout in which A is a set of *m* activation functions. The input is copied to each unit, which is linearly combined with learnable weights and then passed through an activation function to apply a non-linear transformation. In this layer, if multiple activations exist, then each unit does not have the same transformation applied. If A is a set containing a single activation function, it reduces to a traditional feed-forward layer. By applying multiple activation functions in a layer, each computational unit can learn a different mapping of the input features as activation functions are used in different aspects, such as sigmoid for thresholding or ReLU for identity transformations of positive values. As different activation functions can produce a diverse possible range of values, batch normalization is applied after the activation functions to ensure that all the outputs are normalized.

Model Diagram

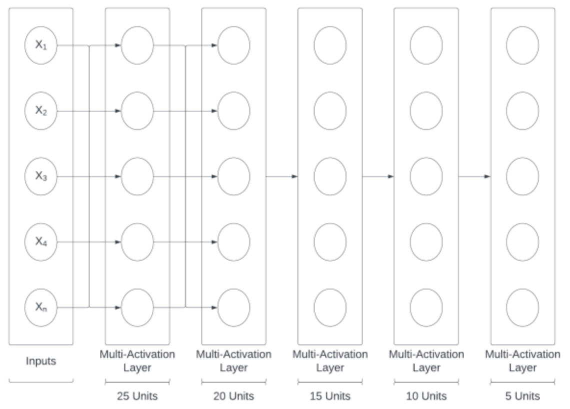

Figure 2- Diagram of the tested model architecture

The model was designed so that the number of activations used would be applied to the same number of units in each layer. For testing, we utilized five activation functions, so every layer contains a decreasing number of units, a multiple of five. The model's output changes depending on the problem type the model was testing with, a single linear regression unit for regression and multiple units with a Softmax activation for classification where the number of units is equal to the number of classes. The model was tested against identical architectures in which each layer only had a single activation function applied to all units to compare the proposed layer with traditional layers. Each layer in the model is fully connected so that the output from a single unit is passed to every unit in the following layer.

Variations

As adding multiple activation functions increases the model complexity, another model was created where dropout was applied after every layer with a probability of 20%, deactivating a fifth of the nodes in the layer during a training iteration. As a fifth of the nodes were deactivated during a training iteration, one of the tested activation functions had a chance to be deactivated, preventing other units from relying on specific combinations of activation functions.

For NLP tasks, another variation was created to add a learnable embedding layer before the hidden layers with a dimension of 50 to convert text into meaningful features specific to the dataset.

Experiments

As the paper is dedicated to testing the performance and the runtime costs for applying different activation functions to each unit in a layer, different datasets are used in the experimentation. The Multi-Activation Layers Neural Network was tested for regression, natural language processing, and classification tasks.

Dataset Description - Regression

The regression dataset contained categorical, numerical, string, and categorical attributes represented as numerical. Posted by and Type of property were categorical, while if the property was under construction, RERA approved, Ready to move, and resale, which were categorical data, were represented as numerical values. Square feet, Number of Rooms, Longitude, and latitude were numerical values, and the property's address was represented as a string. The target value labeled "target price" was a numerical value. The dataset did not contain any null, NaN, or missing values.

The dataset used for regression was based on a house price prediction. The openly available dataset had already been split into training and testing datasets. As the testing dataset did not have any price label available, the training dataset available was used for the training and testing. The training dataset consisted of 29451 observations and was split into training and testing datasets using the split of eighty to twenty percent, respectively.

The numerical values of the number of rooms, square feet, and the target price ranged from 1 to 20, 3 square feet to 254545454.5 square feet, and twenty-five thousand to Three hundred million rupees, respectively. Similarly, the categorical attributes, and categorical attributes represented as numerical, showcased similar forms of data distribution imbalance in the dataset. The attribute "Posted by" had a categorical distribution of Dealer, Owner, and Builder, sixty-two, thirty-six, and two percent, respectively. The data showcased that only eighteen percent of the buildings were under construction, only thirty-two percent of the buildings were RERA approved, yet eighty-two percent of the buildings were ready to be moved into, and ninety-three percent of the buildings were marked for resale.

A one-hot encoder and a scaler transform were implemented to clean and normalize the dataset, and the address column was removed. This allowed the dataset to homogenize and be represented as numerical values, making it easier to train for all machine learning methods, as they would only have to deal with a single data type.

Dataset Description - Classification

The dataset used for the classification dataset contained 918 observations and were split into 80 and 20 percent, respectively, for training and testing, similar to the regression dataset. The dataset contained varying data types ranging from categorical, categorical numerical, numerical, and labels.

The numerical data attributes in the dataset consisted of Age, Resting blood pressure, Cholesterol, Maximum heart rate, and Oldpeak, represented as the numerical value for ST. ST is the numerical value measured in depression. The categorical data attributes consisted of Sex, Chest Pain Type, Resting ECG, Exercise Angina, and ST Slope. At the same time, the numerical data for the fasting blood sugar was presented as a category; the categorical target values of if the patient had heart disease or was normal were represented numerically.

Similar to the regression dataset, the classification dataset varied in range and distribution per column. The numerical values for Age ranged from 28 to 77, with a median of 53.5 and a standard deviation of 9.43. Within the 918 data observations, 79% of the population was represented by males. The resting blood pressure ranged from 80 mm Hg to 200 mm Hg, with a median of 132 and a standard deviation of 18.5. The highest data variation was present in Cholesterol, as the data ranged from 0 to 603. As nearly 19% of the dataset observations consisted of the total serum cholesterol ranging from 0 to 60, the data for the cholesterol remained untouched. Similarly to the regression dataset, one hot encoder and scaler transform were implemented to clean and normalize the dataset. This allowed the dataset to homogenize and be represented as numerical values.

Dataset Description – Natural Language Processing

The dataset used for Natural language processing contained over 7800 observations in string format. The dataset contained search results of posts that describe the personality of Twitter users. The dataset used for testing ensured the dataset's quality by only including users who shared more than 200 words. (Reference to Kaggle for this).

Techniques such as vector embedding and limitation were implemented to preprocess the dataset. To further clean the dataset, the dataset was cleared of any stop words, punctuation, and special characters and symbols such as but not limited to "|,.@#."

Test Results and Evaluation

The datasets were trained and tested using 5 Uniform Activation, 1 Sequential, 1 Multi-Activation, and 1 Multi-Activation with dropout Feed Forward Neural Network models. Each machine-learning model was fitted to the training data using 250 epochs with a batch size of 128. The results shown ahead represent the average accuracy and loss rate over five executions of each model. The performance of the proposed architecture was compared with the traditional machine learning models as a baseline.

Throughout training and testing the models, the proposed multi-activation architecture with dropout performed the slowest in terms of training loss overall on average but performed better during some iterations.

Regression – Results

The regression dataset's average root mean squared error over five runtime iterations was used as the performance evaluator.

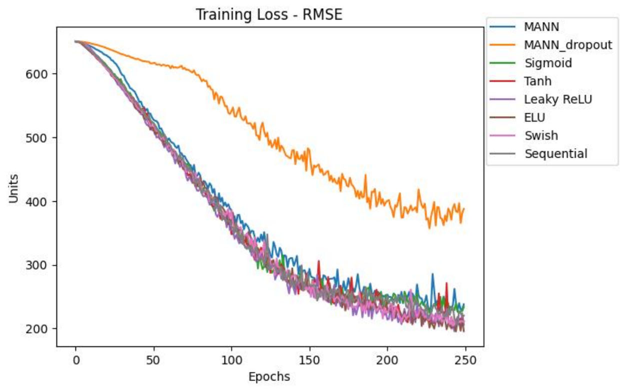

Figure 3 – Regression – Training loss – RMSE

As seen in Figure 3, although the training loss for the proposed architecture with dropout decreases slower than the first 100 epochs, it tends to follow a similar trend of training loss as all the other models. Alternatively, the proposed architecture does as well as other training loss models.

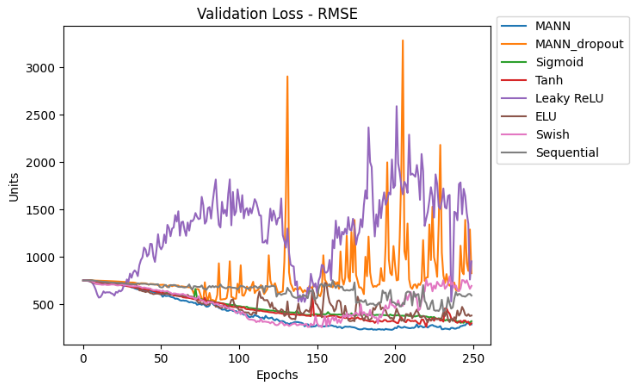

Figure 4 – Regression – Validation loss – RMSE

Similar to training loss, the proposed architecture with dropout does not do well compared to other baseline machine learning methods, yet surprisingly the sequential model does equally as poorly.

Model	Root Mean Squared Error
Tanh NN	306.924344
Sigmoid NN	344.552533
Swish NN	364.056186
MANN	479.159814
ELU NN	497.860382
MANN w/ Dropout	520.247321
Sequential NN	530.127789
Leaky ReLU NN	539.898950
Linear Regression	571.108276
Support Vector Machine	586.254749

Table 1 – Regression – Testing RMSE

Based on the data in Table 1 and Figures 3 and 4, the proposed architecture performs nearly as well as all the other uniform models in terms of loss and Root Mean Squared Error. Surprisingly over the average of 5 different iterations, it outperforms Linear Regression and Support Vector Machines.

Classification – Results

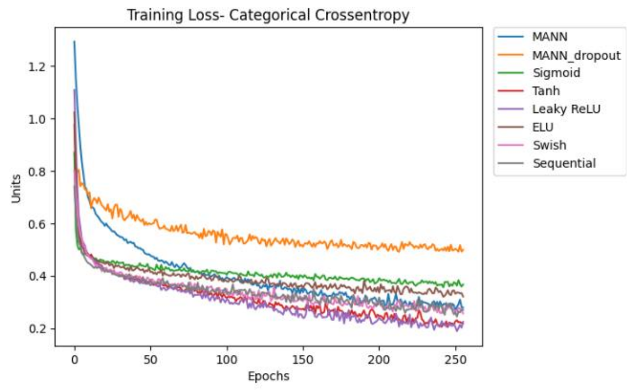

Figure 6 – Classification – Training Loss

As seen in Figure 3, the proposed architecture with dropout follows a similar trend for training loss. Although it does not achieve the same training loss as all the other models, it is just as stable as all other models in Figure 6.

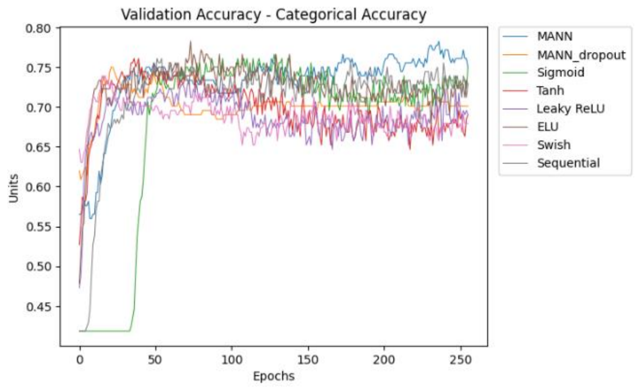

Figure 7 – Classification – Validation Accuracy

Although the Multi-Activation model with dropout cannot achieve similar scores during training for loss as all other models, it is one of the two models that can achieve high accuracy from the beginning. Further testing with higher observations and different dataset complexity may be required to judge the multi-Activation model with dropout's performance for classification accurately. Comparatively, the proposed architecture of the multi-Activation model performs the best during the validation accuracy.

Model	Categorical Accuracy (%)
Logistic Regression	73.913
MANN w/ Dropout	72.4185
Sigmoid NN	72.1467
MANN	71.875
ELU NN	71.1957
Tanh NN	70.7881
Swish NN	69.7011
Sequential NN	68.8859
Leaky ReLU NN	66.712

Table 2 – Classification – Testing Accuracy

Surprisingly, although the Multi-Activation model with dropout may not perform the best in training loss and validation accuracy, over multiple iterations of testing and training the model, it achieves the second-highest categorical accuracy, arriving at

1.5% short of Logistic Regression while performing nearly as well as Sigmoid.

NLP - Results

Based on the results presented in Figure 9, nearly all the models besides Multi-Activation, and Multi-activation with dropout and tanh reach a training loss of 0 within less than 50 epochs. According to the training loss, Tanh is entirely unable to learn from the dataset; all the while, the proposed architectures show better results than other models. The models may be overfitting the dataset, as there may be too many unique words present, causing it to be unable to learn properly.

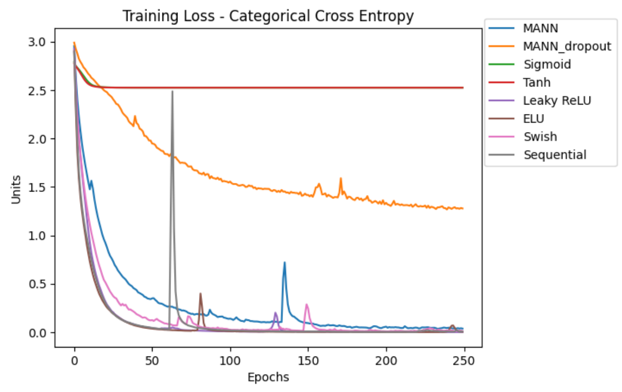

Figure 9 – NLP – Training Loss

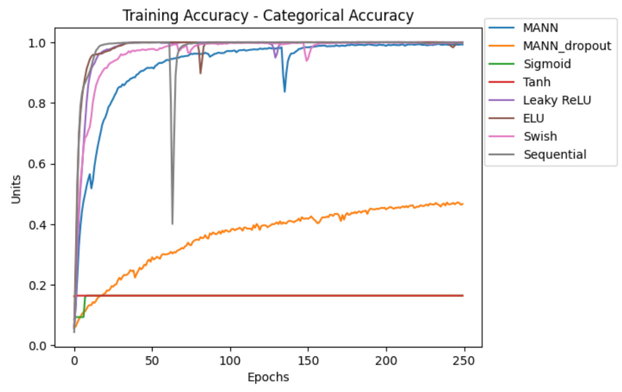

Figure 10 – NLP – Training Accuracy

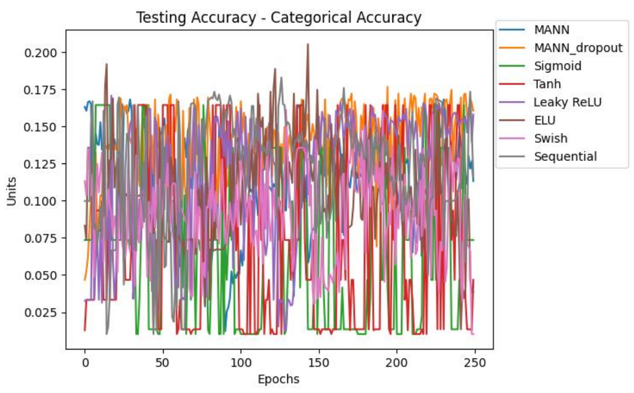

Figure 11 – NLP – Testing Accuracy

Based on Figure 11 and Table 3, further testing is required for the Natural language processing data.

Model	Categorical Accuracy (%)
Logistic Regression	26.4875
Naive Bayes	21.8170
MANN w/ Dropout	16.0589
MANN	14.7153
Leaky ReLU NN	13.0518
Sigmoid NN	10.3647
Swish NN	9.7889
Tanh NN	7.3576
Sequentail NN	7.3576
ELU NN	4.3506

Table 3 – NLP – Training Accuracy

Conclusions

The model with the proposed layer architecture achieved approximately similar performance results and, in some iterations of training, outperformed the identical architectures with a single activation per layer. While the proposed architecture performed about the same as other models, an important observation is that the model, on average, converged faster on the optimal weights for testing data than the other tested models. This implies that our proposed architecture requires much fewer epochs than applying a single activation function for each layer to generalize and achieve similar performance.

Further improvements, such as parameter tuning and generalizations, can be made to the proposed architecture by testing different datasets with varying feature counts. Additional testing can be done to analyze the weights for each unit in the model to determine which activation functions are primarily used in each layer. This approach could allow the proposed architecture to serve as an exploratory tool to determine the optimal sequence of activation functions for a given model design.

References

Anand, R., Mehrotra, K., Mohan, C. K., & Ranka, S. (1995). Efficient classification for multiclass problems using modular neural networks. *IEEE transactions on Neural Networks*, *6*(1), 117-124.

Bishop, Chris M., "Neural networks and their applications". *Review of Scientific Instruments* 65. 6(1994): 1803-1832. https://doi.org/10.1063/1.1144830

Ding, B., Qian, H., & Zhou, J. (2018). Activation functions and their characteristics in deep neural networks. *2018 Chinese Control And Decision Conference (CCDC).* https://doi.org/10.1109/ccdc.2018.8407425

Ronco, Eric, and Gawthrop, Peter. "Modular neural networks: a state of the art." *Rapport Technique CSC-95026, Center of System and Control, University of Glasgow. http://www. mech. gla. ac. uk/control/report. html* (1995).

Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, *04*(12), 310–316. https://doi.org/10.33564/ijeast.2020.v04i12.054

Vaswani, A., et al. "Attention is All you Need." *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

Vijayakumar, K., Kadam, V. J., & Sharma, S. K. (2021). Breast cancer diagnosis using multiple activation deep neural network. *Concurrent Engineering*, *29*(3), 275–284. https://doi.org/10.1177/1063293x211025105