Learning Activations in NeuralNetworks

***Abstract*—The choice of Activation Functions (AF) has proven to be an important factor that affects the performance of an Artificial Neural Network (ANN). Use a 1-hidden layer neural network model that adapts to the most suitable activation function according to the data-set. The ANN model can learn for itself the best AF to use by exploiting a flexible functional form, $k_0 + k_1 * x$ with parameters k_0, k_1 being learned from multiple runs. You can use this code-base for implementation guidelines and help. https://github.com/sahamath/MultiLayerPerceptron**

I. Background

Selection of the best performing AF for classification task is essentially a naive (or brute-force) procedure wherein, a popularly used AF is picked and used in the network for approximating the optimal function. If this function fails, the process is repeated with a different AF, till the network learns to approximate the ideal function. It is interesting to inquire and inspect whether there exists a possibility of building a framework which uses the inherent clues and insights from data and bring about the most suitable AF. The possibilities of such an approach could not only save significant time and effort for tuning the model, but will also open up new ways for discovering essential features of not-so-popular AFs.

II. Problem Statement

Given a specific activation function

$$g(x) = k_0 + k_1 x \tag{1}$$

and categorical cross-entropy loss, design a Neural Network on Banknote, MNIST or IRIS data where the activation function parameters k_0, k_1 are learned from the data you choose from one of the above-mentioned data sets. solution must include the learnable parameter values i.e. final k_0, k_1 values at the end of training, a plot depicting changes in k_0, k_1 at each epoch, training vs test loss, train vs. test accuracy and a Loss function plot.

III. Mathematical Framework

A. Compact Representation

Let the proposed Ada-Act activation function be mathematically defined as:

$$g(x) = k_0 + k_1 x \tag{2}$$

where the coefficients k_0, k_1 have to be learned during training via back-propagation of error gradients, on a particular data set specified in the problem statement.

For the purpose of demonstration, consider a feed-forward neural network consisting of an input layer L_0 consisting of m nodes for m features, two hidden layers L_1 and L_2 consisting of n and p nodes respectively, and an output layer L_3 consisting of k nodes for k classes. Let z_i and a_i denote the inputs to and the activations of the nodes in layer L_i respectively. Let w_i and b_i denote the weights and the biases applied to the nodes of layer L_{i-1}, and let the activations of layer L_0 be the input features of the training examples. Finally, let K denote the column matrix containing the equation coefficients: $\begin{bmatrix} k_0 \\ k_1 \\ k_2 \end{bmatrix}$ and let t denote the number of training examples being taken in one batch. Then the forward-propagation equations will be:

$$z_1 = a_0 \times w_1 + b_1$$

$$a_1 = g(z_1)$$

$$z_2 = a_1 \times w_2 + b_2$$

$$a_2 = g(z_2)$$

$$z_3 = a_2 \times w_3 + b_3$$

$$a_3 = Softmax(z_3)$$

where $\times$ denotes the matrix multiplication operation and $Softmax()$ denotes the Softmax activation function.

For back-propagation, let the loss function used in this model be the Categorical Cross-Entropy Loss, and let df_i denote the gradient matrix of the loss with respect to the matrix f_i, where f can be substituted with z, a, b, or w. and let there be matrices dK_2 and dK_1 of dimension 3×1. Then the back-propagation equations will be:

$$dz_3 = a_3 - y$$

$$dw_3 = \frac{1}{t} a_2^T \times dz_3$$

$$db_3 = avg_{\text{col}}(dz_3)$$

$$da_2 = dz_3 \times w_3^T$$

$$dz_2 = g'(z_2) * da_2$$

$$dw_2 = \frac{1}{t} a_1^T \times dz_2$$
$$db_2 = avg_{\text{col}}(dz_2)$$

$$da_1 = dz_2 \times w_2^T$$
$$dz_1 = g'(z_1) * da_1$$
$$dw_1 = \frac{1}{t} a_0^T \times dz_1$$
$$db_1 = avg_{\text{col}}(dz_1)$$

$$dK_1 = \begin{bmatrix} avg_e(da_1) \\ avg_e(da_1 * z_1) \\ avg_e(da_1 * z_1^2) \end{bmatrix} \tag{3}$$
$$dK = dK_1 \tag{4}$$

where $*$ is the element-wise multiplication operation, T is the matrix transpose operation, $avg_{\text{col}}(x)$ is the function which returns the column-wise average of the elements present in the matrix x, and $avg_e(x)$ is the function which returns the average of all the elements present in the matrix x.

Consider the learning rate of the model to be α. The update equations for the model parameters will be:

$$w_1 = w_1 - \alpha.dw_1$$
$$b_1 = b_1 - \alpha.db_1$$
$$w_2 = w_2 - \alpha.dw_2$$
$$b_2 = b_2 - \alpha.db_2$$
$$w_3 = w_3 - \alpha.dw_3$$
$$b_3 = b_3 - \alpha.db_3$$
$$K = K - \alpha.dK$$