**Self-Learning Activation Functions Neural Network**

**by:-Prashant Sharma**

**Github:https://github.com/prashant-py-debug/SLAF**

**Objective:** Instead of using pre-defined Activation functions we trying to make the NN learn the activation function for itself during training.

**Background:** Selection of the Best-performing Activation Functions for classification tasks are essentially dependent on the background knowledge and experience level. Thus, this involves extra layer of complexity to the already complex process of hyper parameter tunning cycle. Use of Self-Learning AFs can reduce the complexity and can develop new optimum AFs for every application.

**Plan:** We are using the Tensorflow 2.0 framework for the gradient calculation, we built a 2-hidden layer NN where First layer is for learning parameters W(for four features ) and B and Second layer is for learning parameters for our generalised Activation function,i.e

y = k0 + k1.x

and in the output layer we used sigmoid AF for the Binary Classification task.

**Dataset**: The dataset we used to train our model is from Kaggle dataset "Bank-Note Authentication". Our dataset contains 1372 records with four Feature Columns and One label column. we examined the dataset and found it clean and balanced. We split the dataset into training and testing dataset with our training set containing 1200 records and test containing rest of the records.

**Execution**: We have used the *Sequential API* to create our vannila model of 3 dense layers with a neuron.

Model: "SLAF\_Net"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

FC\_layer\_1 (Dense) (None, 1) 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Activation\_layer (Dense) (None, 1) 2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Output\_layer (Dense) (None, 1) 2

=================================================================

**Initialization**: We have used *Xavier uniform initializer* method to initialise our parameters. Our learning rate is set at 0.001 and we have used the threshold value of 60%. We trained our model for 200 Epochs and used the Batch size of 16 records.

**Training**: We have used *Tensorflow's Dataset API* to feed our data to the model and then used the *BinaryCrossEntropy* loss metric for our calculating Loss. after that used *Tensorflow's GradientTape* for calculating gradients of Loss func with respect to trainable parameters in Backpropagation step. Finally we used *SGD optimizer* for updatation of parameter.

Our training took around 9 minutes on 8 thread CPU and we have witnessed considerable reduction in our Loss and significant gain in accuracy and F1 score both on train and test data.

**Metrics**:

F1 score: 0.984 Accuracy: 0.989 Loss: 3.042 Test accuracy: 0.9825 Test loss: 0.05252

**Plots:**

Train loss vs test accuracy



Train accuracy vs test loss



Parameter K0 vs Epochs:



Parameter K1 vs Epochs



**Result**: After training our model we found the optimum parameters for our Activation Function:

*K0*: [0.44404453] *k1*: [-1.3430157]