Learning Activations in Neural Network

Deep Dutta
Notebook Link : https://github.com/itzzdeep/Learning-Activation

ABSTRACT

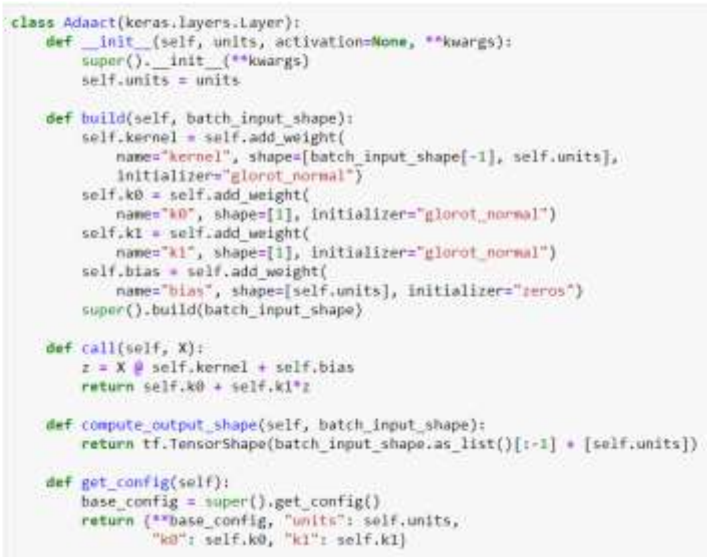
The choice of Activation Functions (AF) has proven to be an important factor that affects the performance of an Artificial Neural Network (ANN). Here I am using a 1-hidden layer neural network model that adapts to the most suitable activation function according to the data-set. The ANN model can learn for itself the best AF to use by exploiting a flexible functional form k0 + k1 * x with parameters k0; k1 being learned from multiple runs.

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.

HIDDEN LAYER

Here I am using the activation function of the functional form g(x) = k0 + k1 * x where the parameter values will be learned from backpropagation. Here, I have done that by subclassing the keras layers class. I have defined a layer where the model will learn the kernel as well as k0 and k1. The Adaact layer looks like following



```
class Adaact(keras.layers.Layer):
    def __init__(self, units, activation=None, **kwargs):
        super().__init__(**kwargs)
        self.units = units

    def build(self, batch_input_shape):
        self.kernel = self.add_weight(
            name="kernel", shape=[batch_input_shape[-1], self.units],
            initializer="glorot_normal")
        self.k0 = self.add_weight(
            name="k0", shape=[1], initializer="glorot_normal")
        self.k1 = self.add_weight(
            name="k1", shape=[1], initializer="glorot_normal")
        self.bias = self.add_weight(
            name="bias", shape=[self.units], initializer="zeros")
        super().build(batch_input_shape)

    def call(self, X):
        z = X @ self.kernel + self.bias
        return self.k0 + self.k1*z

    def compute_output_shape(self, batch_input_shape):
        return tf.TensorShape(batch_input_shape.as_list()[:-1] + [self.units])

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "units": self.units,
                "k0": self.k0, "k1": self.k1}
```

ALGORITHM

The algorithm works in the following way. Given the input we have defined the kernel matrix shape of [d X number of units] where d is the shaped after stretching the unit all the way. The kerel is initialized with glorot initialization. In the same way we have initiated the k0 and k1 with glorot initialization.

Glorot Initialization

Normal distribution with mean 0 and Variance $(1/fan_{avg})$

Where,

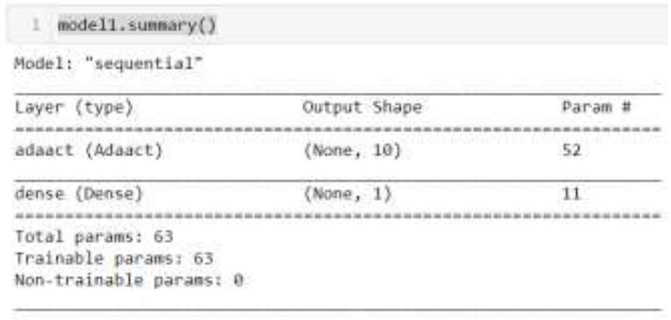
$$fan_{avg} = (fan_{in} + fan_{out}) / 2$$

Also, we have initialized the biased with zeroes. When we will call the model, it will first compute the dot product of kernel and input and adding a bias. Then, it will take the resulted output as the input of the function g(x) and the return the value. The kernel, k0, k1 and bias, all will be learned from backpropagation.

PARAMETERS

We can find the total number of parameters from the model summary and here we will see the different numbers of parameters for three different datasets.

For the Bank Note Dataset, we haven the adaact layer with units. So, the shape of the kernel [4 X 10] i.e. 40 parameters of the kernel and a bias term for each unit which is a total of 10. There is also k0 and k2. So, total number of parameters to be learned is (40 + 10 + 2) = 52. In the second layer we have taken an one unit fc (fully connected) layer with sigmoid non-linearity. The output of the first layer is [1 X 10] and this will be input of the previous layer. So, we have a kernel of shape [10 X 1] i.e. 10 parameter and a bias. So, a total of 11 parameters in second layer. So in this we have a total of 63 parameters in this model



```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
adaact (Adaact)              (None, 10)                52
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================
Total params: 63
Trainable params: 63
Non-trainable params: 0
_________________________________________________________________
```

In the similar way, we can see the parameters for other data sets. For MNIST dataset the first 64 unit adaact layer have a total of 50242 parameters. As every image is of shape [28 X 28]. So, the kernel has a shape of [784 X 64] i.e., 50176 and 64 bias and k0 and k2. So total parameter is 50242.

RESULT

Initial Setting of k0 & k1

Initial k0 and k1 were sampled from normal distribution having mean 0 and variance (1/fan$_{avg}$).

Updates of Parameter on Each epoch & Final Parameter

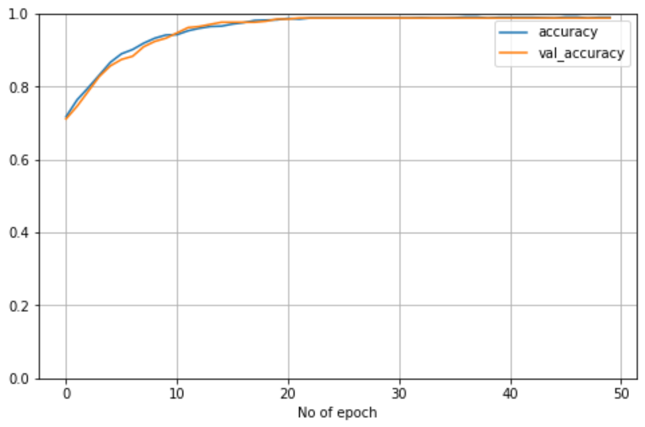
The parameter updates is shown on the notebook after every epoch and also the final parameter is shown separately.

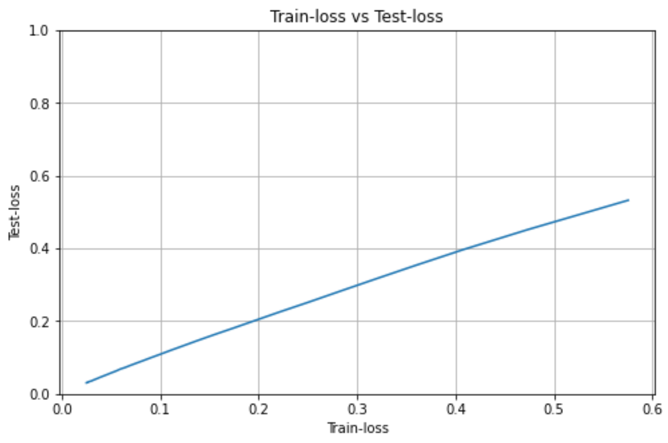
Plots

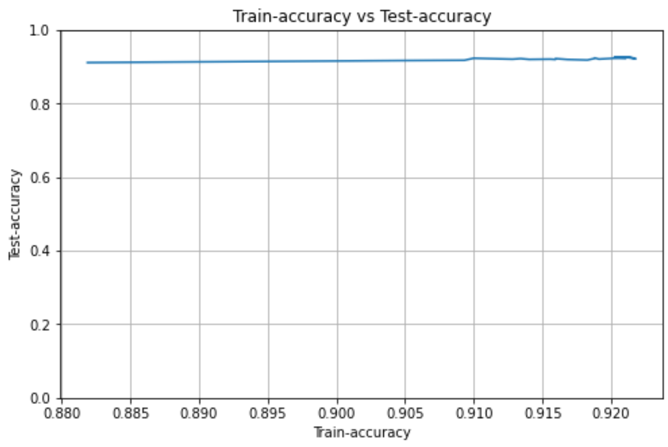
All the plots have been shown in notebook. Also I am showing the plots here also.

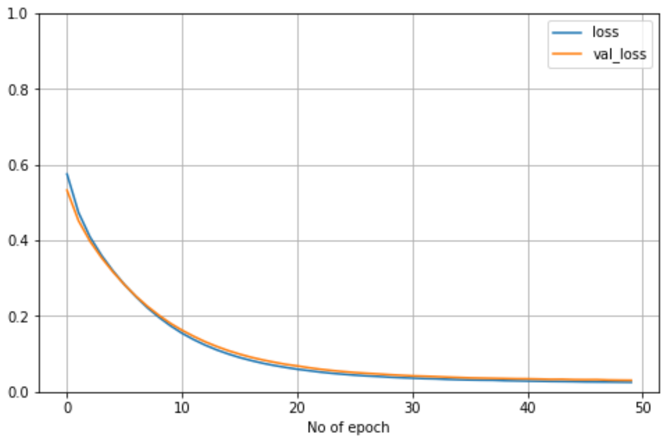
BANK NOTE DATASET

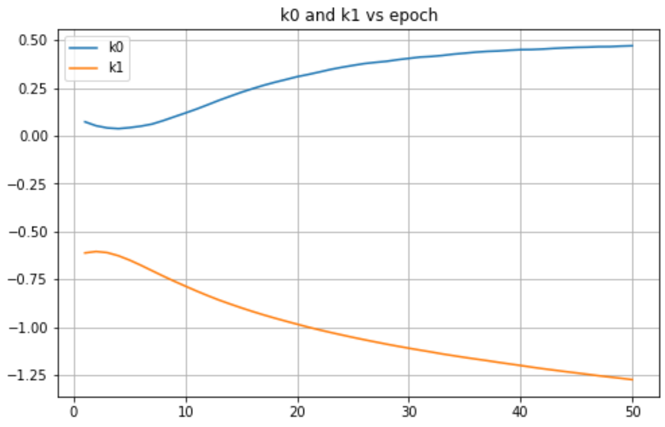
In Bank Note Dataset, the corresponding train and test accuracy is 0.991 and 0.988. The f1 score for the test set is 0.987 and the plots are following





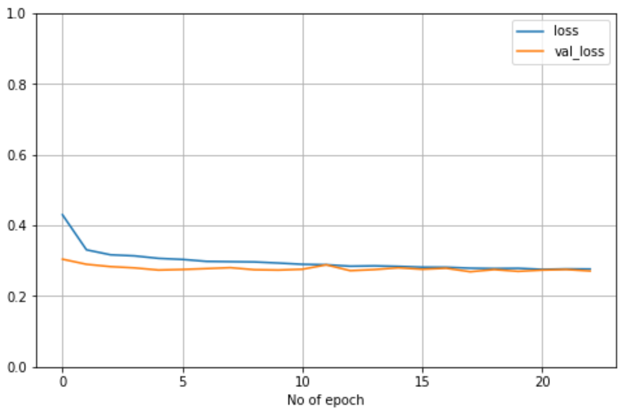


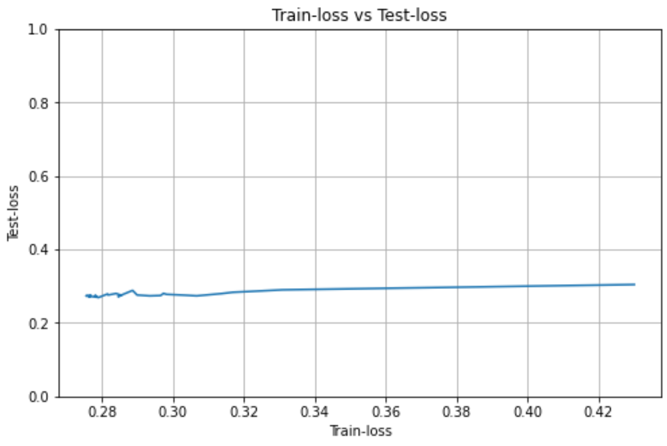


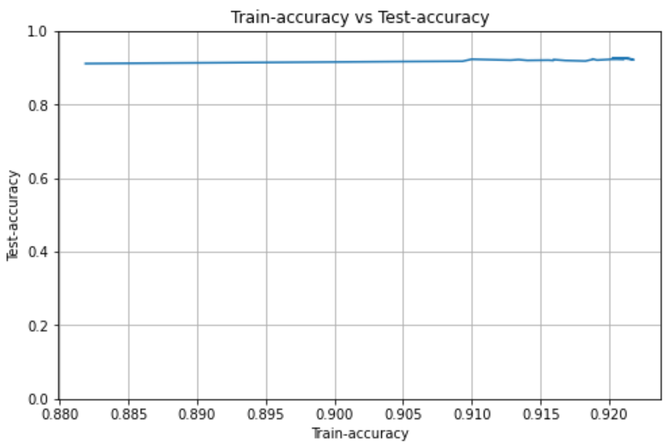


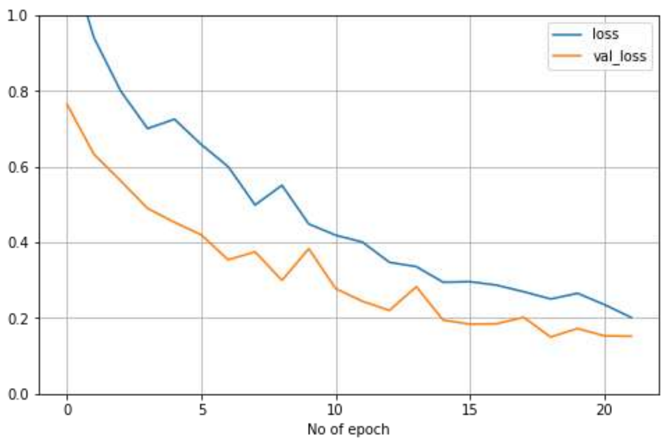
MNIST DATASET

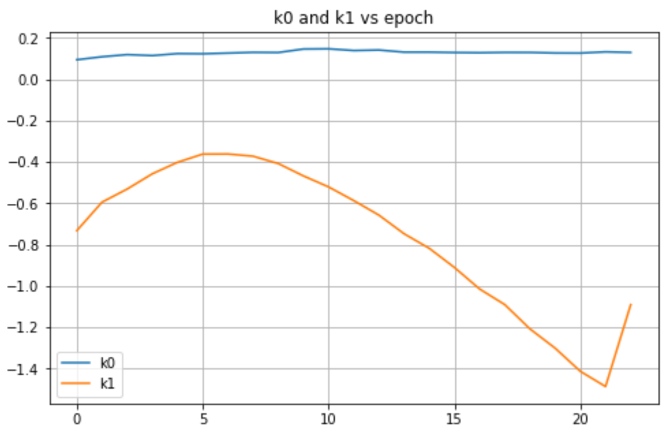
In Bank Note Dataset, the corresponding train and test accuracy is 0.931 and 0.924. The f1 score for the test set is 0.924 and the plots are following

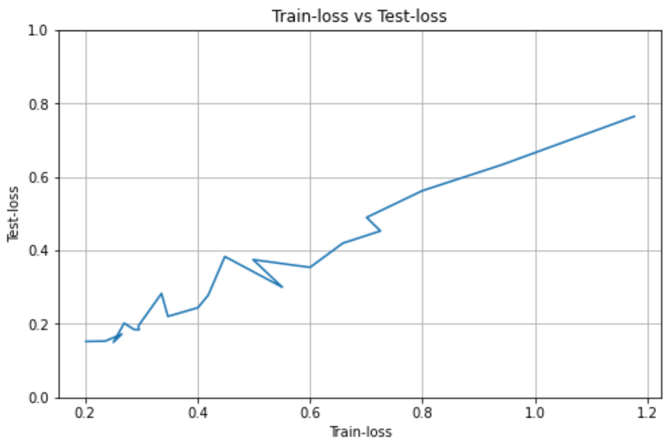


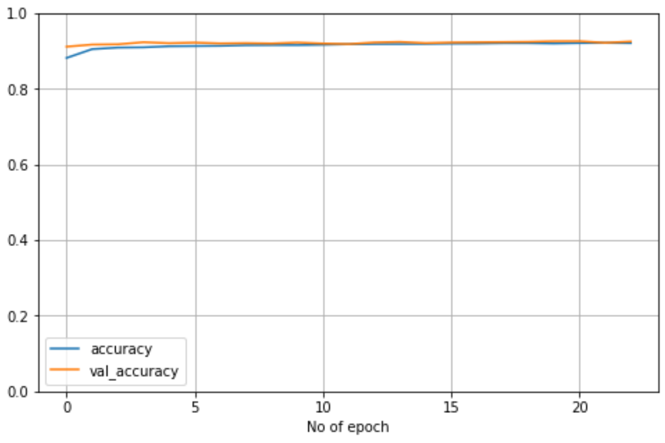


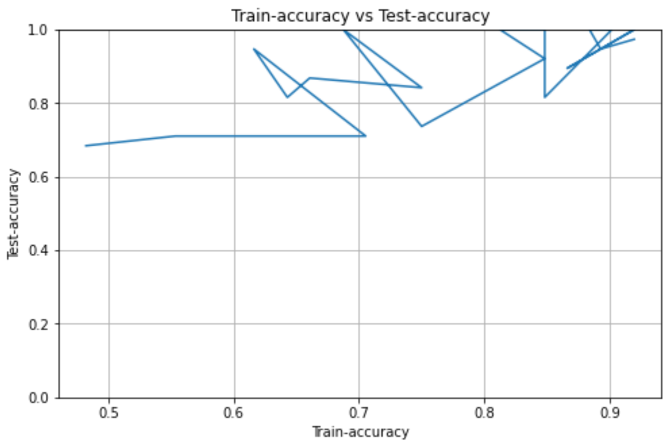






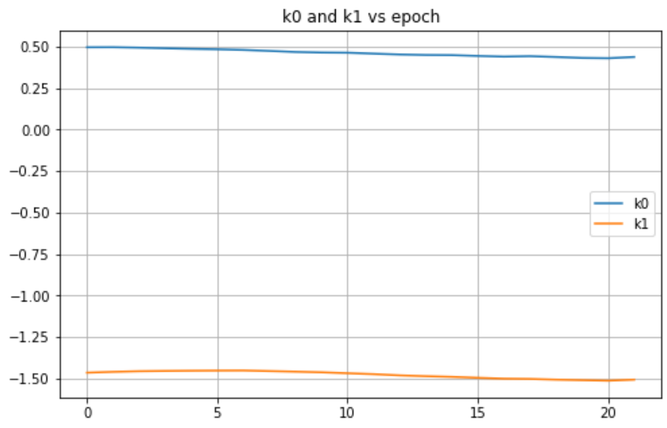


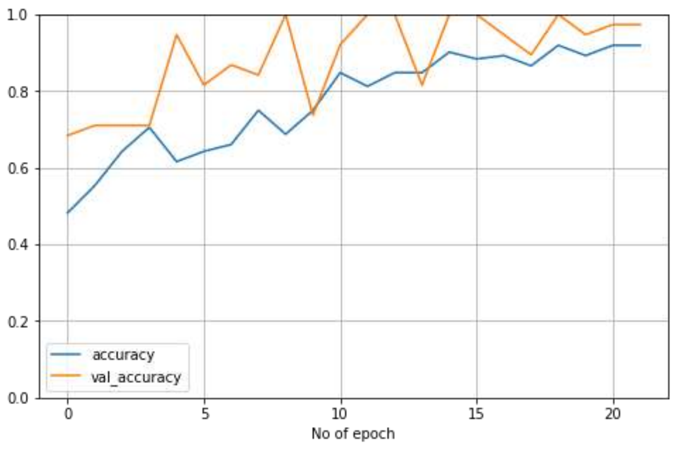




IRIS DATASET

In Bank Note Dataset, the corresponding train and test accuracy is 0.982 and 1.0. The f1 score for the test set is 1.0 and the plots are following


k0 and k1 vs epoch
k0
k1


accuracy
val_accuracy
No of epoch