Study of ensemble of activation functions in Deep Networks

Research Practice Report

Submitted in fulfillment of the requirements of
CS G540
By

Jagannath Kumar(2021H1030065G)
Shirsekar Kshitija Jaykumar(2021H1030052G)

Under the supervision of
Dr. Snehanshu Saha

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI,
K.K BIRLA GOA CAMPUS

May 2022

INDEX

INTRODUCTION

A neural network is a multi-layered network of neurons made up of nodes that are used to classify and predict data provided as input to the network. It has an input layer, one or more hidden layers, and an output layer. Every layer has nodes, and each node has a weight that is taken into account when processing information from one layer to another.

Activation functions make the neural network non-linear i.e. the output depends linearly on the input features. Despite the fact that a linear equation is straightforward and easy to understand, their complexity is limited and they no longer have the potential to examine and recognize complex mappings from the information. A neural network without an activation function acts as a linear regression model with limited performance and power most of the time. It is expected that a neural network not only learns and computes a linear characteristic but performs more complicated responsibilities than modeling complicated types of information consisting of snapshots, movies, audio, speech, text, and many others. This is why we use activation function capabilities and artificial neural network strategies like deep learning that simplify complex, high dimensional, and nonlinear data sets wherein the models have multiple hidden layers for extracting knowledge and useful information.

In simple terms, the activation function decides whether a neuron in a neural network should be activated or not, which is similar to the working of our brain. These functions play an important role in back propagation that is used to update the weights and biases in a neural network.

EXPERIMENTS AND RESULTS

MNIST dataset:

The Modified National Institute of Standards and Technology dataset abbreviated as MNIST dataset consists of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. Our first model experiments using a 2D convolution neural network with alternating MaxPooling layer defined as follows:

```
#load training data and split into train and test sets
mnist = tf.keras.datasets.mnist

(x_train,y_train), (x_test,y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

classes=[0,1,2,3,4,5,6,7,8,9]

model1 = tf.keras.models.Sequential()
model1.add(tf.keras.layers.Conv2D(32,(3,3),activation='relu',inp
ut_shape=(28, 28, 1)))      #Convo2D 1st(input layer) in table
model1.add(tf.keras.layers.MaxPooling2D((2, 2)))
model1.add(tf.keras.layers.Conv2D(64,(3,3),activation='relu'))
#Convo2D 2nd in table
model1.add(tf.keras.layers.MaxPooling2D((2, 2)))
model1.add(tf.keras.layers.Conv2D(64,(3,  3),  activation='relu'))
#Convo2D 3rd in table
model1.add(tf.keras.layers.Flatten())
model1.add(tf.keras.layers.Dense(64,  activation='relu'))  #Dense1
in table
model1.add(tf.keras.layers.Dense(10, activation='softmax'))

#train the model
model1.fit(x_train,y_train,epochs=9,validation_data=(x_test,y_te
st))
```

The results obtained by changing the activation functions in various layers are shown in the images below:

Convo2D 1st(input layer) / Rest	ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu	99.25 , 2.76	98.85 , 4.41	99.13 , 2.93	99.01 , 3.45	99.13 , 4.12	99.1 , 3.86
LeakyRelu	99.01 , 3.4	99.2 , 3.07	98.79 , 4.13	99.08 , 3.21	99.0 , 4.35	99.14 , 3.63
Sigmoid	97.84 , 7.12	98.48 , 4.9	99.0 , 3.19	98.56 , 4.55	98.89 , 4.7	98.87 , 4.05
Tanh	99.09 , 3.20	98.86 , 4.25	99.16 , 2.55	98.89 , 4.16	99.12 , 3.84	98.94 , 5.73
Swish	99.24 , 2.98	98.9 , 4.79	99.02 , 2.77	98.95 , 3.44	98.97 , 4.18	99.28 , 3.28
Mish	99.06 , 3.58	99.01 , 3.07	99.13 , 2.81	98.76 , 3.93	99.29 , 3.31	98.99 , 4.27

Fig.1- MNIST result 1

Here the output is the correct classification of the handwritten numbers in the ten classes from 0 to 9. The table shows what activation function was used in the first Convolution layer and what was used in the other three layers(Conv2D 2nd, 3rd, and Dense1). Results obtained after such combinations are displayed in the corresponding cell where green signifies the percentage of validation accuracy and red signifies the percentage of validation loss.

Similarly, the other three results are displayed further:

Convo2D 2nd / Rest	ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu	99.25 , 2.76	99.22 , 3.31	98.57 , 4.44	98.78 , 3.83	99.12 , 3.58	98.94 , 4.42
LeakyRelu	99.1 , 3.4	99.2 , 3.07	98.64 , 4.05	99.05 , 3.44	99.27 , 3.5	98.98 , 4.2
Sigmoid	98.76 , 4.01	98.73 , 4.16	99.0 , 3.19	99.0 , 3.03	99.05 , 3.30	99.0 , 3.58
Tanh	99.27 , 3.08	98.94 , 4.77	98.53 , 4.47	98.89 , 4.16	98.98 , 4.29	98.92 , 4.53
Swish	99.08 , 3.20	98.84 , 4.35	99.01 , 3.07	99.1 , 2.97	98.97 , 4.18	98.84 , 5.12
Mish	99.10 , 3.68	99.21 , 3.26	98.84 , 3.71	99.13 , 2.98	99.18 , 3.62	98.99 , 4.27

Fig.2- MNIST result 2

Rest / Convo2D 3rd	ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu	99.25 , 2.76	99.15 , 3.32	98.88 , 3.5	98.96 , 3.53	99.14 , 3.68	98.99 , 4.84
LeakyRelu	98.97 , 3.93	99.2 , 3.07	98.92 , 3.71	99.18 , 3.14	99.16 , 3.97	98.92 , 4.10
Sigmoid	99.03 , 3.32	98.9 , 3.62	99.0 , 3.19	99.0 , 3.79	98.87 , 4.44	99.21, 2.75
Tanh	98.94 , 4.34	98.64 , 5.74	98.78 , 3.76	98.89 , 4.16	99.17 , 3.33	98.87 , 5.07
Swish	98.95 , 4.26	98.98 , 3.48	98.86 , 3.51	99.07 , 3.57	98.97 , 4.18	99.08 , 3.75
Mish	99.17 , 3.97	99.19 , 3.07	98.85 , 3.92	98.96 , 3.82	99.0 , 4.15	98.99 , 4.27

Fig.3 - MNIST result 3

Rest / Dense 1	ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu	99.25 , 2.76	99.09 , 3.6	98.98 , 3.35	98.98 , 4.55	98.66 , 5.58	99.03 , 4.16
LeakyRelu	99.15 , 3.17	99.2 , 3.07	98.87 , 3.77	98.86 , 4.59	99.19 , 3.33	98.91 , 5.26
Sigmoid	99.04 , 3.01	99.1 , 3.06	99.0 , 3.19	98.83 , 4.12	99.01 , 3.39	99.23 , 2.78
Tanh	99.28 , 2.59	99.24 , 2.69	99.06 , 3.26	98.89 , 4.16	99.0 , 4.01	99.23 , 3.4
Swish	99.2 , 3.03	99.04 , 3.54	98.8 , 3.79	98.88 , 4.78	98.97 , 4.18	99.06 , 3.78
Mish	99.13 , 3.69	99.19 , 3.47	98.77 , 3.9	98.93 , 4.69	99.23 , 3.23	98.99 , 4.27

Fig.4 - MNIST result 4

CIFAR-100 dataset:

The CIFAR-100 dataset (Canadian Institute for Advanced Research, 100 classes) is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. There are 600 images per class. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). There are 500 training images and 100 testing images per class. We built a CNN model that can correctly recognize and classify colored images of objects into one of the 100 available classes of the CIFAR-100 dataset. Here is the list of classes in the CIFAR-100:

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Test accuracy for a single layer with one activation function is:

Activation Function	Accuracy
Relu	0.68
Prelu	0.70
Tanh	0.67
Elu	0.70
Selu	0.71
Mish	0.69
Swish	0.68

Fig.4 - CIFAR100 result 1

For a combination of activation functions in 3 layers the results are as follows:

```
(x_train, y_train), (x_test, y_test) =
keras.datasets.cifar100.load_data()
#initializing CNN model
model = Sequential()
#convolution
```

```
model.add(Conv2D(filters=32, kernel_size=3, padding="same",
activation="relu", input_shape=x_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=3, padding="same",
activation="relu")) #layer1 in table
#pooling
model.add(MaxPool2D(pool_size=2, strides=2))
#dropout
model.add(Dropout(0.2))
#stack2
#convolution
model.add(Conv2D(filters=64, kernel_size=3, padding="same",
activation="relu"))
model.add(Conv2D(filters=64, kernel_size=3, padding="same",
activation="relu")) #layer2 in table
model.add(MaxPool2D(pool_size=2, strides=2))
#dropout model.add(Dropout(0.5))
#Stack 3
#convolution
model.add(Conv2D(filters=96, kernel_size=3, padding="same",
activation="relu"))
model.add(Conv2D(filters=96, kernel_size=3, padding="same",
activation="relu"))  #layer3 in table
model.add(MaxPool2D(pool_size=2, strides=2))
model.add(Dropout(0.5))
#flattening
model.add(Flatten())
model.add(Dense(units=100, activation="relu"))
model.add(Dropout(0.5)
model.add(Dense(units=100, activation="relu"))
model.add(Dropout(0.5))
#output layer
model.add(Dense(units=100, activation="softmax"))
```

layer1(32),layer2(64),layer3(64)	Accuracy,test error
1. tanh, swish, tanh	0.66 0.96
2. tanh, tanh, swish	0.68 0.93
3. Tanh , swish,swish	0.68 0.92
4. Tanh , tanh , tanh	0.66 0.96
5. tanh, relu,sigmoid	0.63 1.04
6. tanh,sigmoid ,relu	0.39 1.62
7.relu , tanh ,sigmoid	0.61 1.10
8. Relu, sigmoid,tanh	0.57 1.20
9. relu,relu,relu	0.64 1.01
10.sigmoid,sigmoid,sigmoid	0.45 1.51
11.sigmoid, tanh,relu	0.50 1.40
12.sigmoid, relu ,tanh	0.10 2.30
13.relu ,sigmoid,sigmoid	0.55 1.23
14.swish,relu,sigmoid	0.59 1.16

Fig.5 - CIFAR100 result 2

BOSTON housing dataset:

This dataset has been built from information collected by the U.S. Census Service concerning housing in the area of Boston Mass. One data frame has 506 rows and 14 columns. This data frame contains the following columns: CRIM per capita(crime rate by town). There are 14 attributes in each case of the dataset. They are:

1. CRIM - per capita crime rate by town
2. ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS - proportion of non-retail business acres per town.
4. CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. NOX - nitric oxides concentration (parts per 10 million)
6. RM - average number of rooms per dwelling
7. AGE - proportion of owner-occupied units built prior to 1940
8. DIS - weighted distances to five Boston employment centres
9. RAD - index of accessibility to radial highways
10. TAX - full-value property-tax rate per $10,000
11. PTRATIO - pupil-teacher ratio by town
12. B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT - % lower status of the population
14. MEDV - Median value of owner-occupied homes in $1000's

```
#load training data and split into train and test sets
(x_train,y_train),(x_test,y_test)=
tf.keras.datasets.boston_housing.load_data(path="boston_housing.
npz", test_split=0.2, seed=113)
```

Dense1 / Dense2	ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu	21.42 , 2.76	17.67 , 2.71	22.98 , 3.51	15.82 , 2.87	16.4 , 2.77	17.62 , 2.66
LeakyRelu	19.17 , 2.73	21.37 , 2.81	22.68 , 3.47	14.42 , 2.61	17.42 , 2.69	16.92 , 2.71
Sigmoid	30.13 , 3.43	26.68 , 3.26	30.78 , 3.67	23.53 , 3.13	27.32 , 3.37	32.27 , 3.53
Tanh	17.13 , 2.85	16.78 , 2.77	21.79 , 3.18	15.38 , 2.77	17.38 , 2.83	16.63 , 2.76
Swish	20.51 , 2.86	16.08 , 2.64	22.73 , 3.49	14.59 , 2.64	15.93 , 2.77	17.35 , 2.67
Mish	18.88 , 2.74	17.47 , 2.66	23.38 , 3.5	15.07 , 2.82	17.41 , 2.69	19.05 , 2.7

Fig.6 -Boston housing result with A.F. in 2 layers

The blue numbers indicate the mean squared error(MSE) and the red numbers in a cell indicate the mean absolute error(MAE).

We have also performed an experiment of permuting activation functions in different consecutive hidden layers of the boston dataset. Its result are given further:

Layer1	Layer2	Layer3	MSE	MAE		Layer1	Layer2	Layer3	MSE	MAE
ReLu	LeakyRelu	Sigmoid	35.58	3.52		LeakyRelu	ReLu	Sigmoid	25.39	3.11
ReLu	LeakyRelu	Tanh	15.58	2.69		LeakyRelu	ReLu	Tanh	16.67	2.67
ReLu	LeakyRelu	Swish	17.93	2.63		LeakyRelu	ReLu	Swish	13.81	2.41
ReLu	LeakyRelu	Mish	16.22	2.59		LeakyRelu	ReLu	Mish	13.5	2.43
ReLu	Sigmoid	LeakyRelu	16.4	2.56		LeakyRelu	Sigmoid	ReLu	16.65	2.57
ReLu	Sigmoid	Tanh	17.93	2.87		LeakyRelu	Sigmoid	Tanh	16.92	2.73
ReLu	Sigmoid	Swish	15.66	2.56		LeakyRelu	Sigmoid	Swish	17.01	2.73
ReLu	Sigmoid	Mish	17.17	2.53		LeakyRelu	Sigmoid	Mish	18.2	2.63
ReLu	Tanh	LeakyRelu	16.84	2.54		LeakyRelu	Tanh	ReLu	15.35	2.53
ReLu	Tanh	Sigmoid	28.27	3.22		LeakyRelu	Tanh	Sigmoid	29.95	3.4
ReLu	Tanh	Swish	12.57	2.39		LeakyRelu	Tanh	Swish	13.72	2.39
ReLu	Tanh	Mish	14.66	2.51		LeakyRelu	Tanh	Mish	15.69	2.5
ReLu	Swish	LeakyRelu	13.15	2.41		LeakyRelu	Swish	ReLu	15.31	2.4
ReLu	Swish	Sigmoid	33.36	3.54		LeakyRelu	Swish	Sigmoid	27.35	3.2
ReLu	Swish	Tanh	15.58	2.6		LeakyRelu	Swish	Tanh	16.78	2.75
ReLu	Swish	Mish	15.76	2.64		LeakyRelu	Swish	Mish	16.14	2.62
ReLu	Mish	LeakyRelu	13.75	2.42		LeakyRelu	Mish	ReLu	19	2.84
ReLu	Mish	Sigmoid	31.7	3.38		LeakyRelu	Mish	Sigmoid	31.8	3.39
ReLu	Mish	Tanh	14.26	2.47		LeakyRelu	Mish	Tanh	15.04	2.39
ReLu	Mish	Swish	13.05	2.51		LeakyRelu	Mish	Swish	16.29	2.55

Layer1	Layer2	Layer3	MSE	MAE		Layer1	Layer2	Layer3	MSE	MAE
Sigmoid	ReLu	LeakyRelu	29.25	3.37		Tanh	ReLu	LeakyRelu	15	2.65
Sigmoid	ReLu	Tanh	20.79	3.07		Tanh	ReLu	Sigmoid	27.7	3.33
Sigmoid	ReLu	Swish	21.16	3.07		Tanh	ReLu	Swish	13.6	2.47
Sigmoid	ReLu	Mish	25.72	3.28		Tanh	ReLu	Mish	15.28	2.46
Sigmoid	LeakyRelu	ReLu	26.06	3.49		Tanh	LeakyRelu	ReLu	15.25	2.69
Sigmoid	LeakyRelu	Tanh	21.77	3.11		Tanh	LeakyRelu	Sigmoid	37.1	3.72
Sigmoid	LeakyRelu	Swish	26.93	3.33		Tanh	LeakyRelu	Swish	12.8	2.41
Sigmoid	LeakyRelu	Mish	29.58	3.58		Tanh	LeakyRelu	Mish	12.93	2.53
Sigmoid	Tanh	ReLu	25.12	3.27		Tanh	Sigmoid	ReLu	17.23	2.79
Sigmoid	Tanh	LeakyRelu	24.38	3.21		Tanh	Sigmoid	LeakyRelu	19.27	2.84
Sigmoid	Tanh	Swish	25.6	3.34		Tanh	Sigmoid	Swish	16.89	2.67
Sigmoid	Tanh	Mish	25.63	3.29		Tanh	Sigmoid	Mish	17.97	2.75
Sigmoid	Swish	ReLu	29	3.46		Tanh	Swish	ReLu	12.53	2.4
Sigmoid	Swish	LeakyRelu	26.52	3.35		Tanh	Swish	LeakyRelu	15.05	2.49
Sigmoid	Swish	Tanh	21.86	3.23		Tanh	Swish	Sigmoid	28.07	3.37
Sigmoid	Swish	Mish	29.2	3.42		Tanh	Swish	Mish	11.04	2.38
Sigmoid	Mish	ReLu	29.22	3.5		Tanh	Mish	ReLu	10.26	2.31
Sigmoid	Mish	LeakyRelu	27.29	3.35		Tanh	Mish	LeakyRelu	13.64	2.62
Sigmoid	Mish	Tanh	21.89	3.16		Tanh	Mish	Sigmoid	34.38	3.59
Sigmoid	Mish	Swish	25	3.18		Tanh	Mish	Swish	13.16	2.55

Layer1	Layer2	Layer3	MSE	MAE		Layer1	Layer2	Layer3	MSE	MAE
Swish	ReLu	LeakyRelu	13.9	2.39		Mish	ReLu	LeakyRelu	16.1	2.51
Swish	ReLu	Sigmoid	35.09	3.5		Mish	ReLu	Sigmoid	30.9	3.37
Swish	ReLu	Tanh	17.35	2.86		Mish	ReLu	Tanh	16.52	2.75
Swish	ReLu	Mish	12.3	2.4		Mish	ReLu	Swish	16.06	2.71
Swish	LeakyRelu	ReLu	15.01	2.59		Mish	LeakyRelu	ReLu	15.03	2.48
Swish	LeakyRelu	Sigmoid	42.08	3.91		Mish	LeakyRelu	Sigmoid	30.6	3.4
Swish	LeakyRelu	Tanh	16.42	2.68		Mish	LeakyRelu	Tanh	15.21	2.54
Swish	LeakyRelu	Mish	15.18	2.51		Mish	LeakyRelu	Swish	17.76	2.5
Swish	Sigmoid	ReLu	18.59	2.6		Mish	Sigmoid	ReLu	19.63	2.75
Swish	Sigmoid	LeakyRelu	18.77	2.66		Mish	Sigmoid	LeakyRelu	17.29	2.58
Swish	Sigmoid	Tanh	18.65	2.84		Mish	Sigmoid	Tanh	18.27	2.89
Swish	Sigmoid	Mish	18.98	2.64		Mish	Sigmoid	Swish	18.98	2.64
Swish	Tanh	ReLu	14.21	2.41		Mish	Tanh	ReLu	16.26	2.84
Swish	Tanh	LeakyRelu	14.79	2.55		Mish	Tanh	LeakyRelu	13.09	2.33
Swish	Tanh	Sigmoid	28.39	3.37		Mish	Tanh	Sigmoid	26.24	3.24
Swish	Tanh	Mish	13.29	2.44		Mish	Tanh	Swish	13.18	2.38
Swish	Mish	ReLu	17.12	2.54		Mish	Swish	ReLu	15.98	2.64
Swish	Mish	LeakyRelu	11.05	2.32		Mish	Swish	LeakyRelu	12.13	2.59
Swish	Mish	Sigmoid	38.86	3.79		Mish	Swish	Sigmoid	23.92	3.06
Swish	Mish	Tanh	15.94	2.64		Mish	Swish	Tanh	16.63	2.7

Fig.7 -Boston housing result with A.F. in 3 consecutive layers

California Dataset:

This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).An household is a group of people residing within a home. Since the average number of rooms and bedrooms in this dataset are provided per household, these columns may take surprisingly large values for block groups with few households and many empty houses.

```
modelcali = Sequential([Dense(160, kernel_initializer='normal',
activation='sigmoid'),Dropout(0.2),Dense(480,kernel_initializer=
'normal', activation='sigmoid'),Dropout(0.2),
    Dense(25, kernel_initializer='normal', activation=mish),
    Dense(1, kernel_initializer='normal', activation='linear')
  ])

#define optimizer,loss function and evaluation metric
msle = MeanSquaredLogarithmicError()
modelcali.compile(optimizer=Adam(learning_rate=0.01),
             loss=msle,
             metrics=['msle'])

#train the model
modelcali.fit(x_train_scaled.values,y_train.values,epochs=10,bat
ch_size=64,validation_split=0.2)
```

Rest / Dense 1(160)		ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu		0.1262	0.1218	146.3747	50.5286	0.1285	0.1275
LeakyRelu		0.1181	0.1191	58.0007	50.8282	0.1348	0.1298
Sigmoid		0.1246	0.1294	56.2237	50.7357	0.1435	0.1354
Tanh		0.1394	0.1422	58.1219	49.7196	0.1601	0.1519
Swish		0.1217	0.1213	146.3747	49.3183	0.1188	0.1234
Mish		0.1192	0.1186	55.2349	49.1385	0.1377	0.1183

Fig.8 - California result 1

Rest / Dense 2(480)		ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu		0.121	0.1181	55.3696	50.1164	0.1211	0.12
LeakyRelu		0.119	0.1143	56.7578	50.2185	0.1169	0.1242
Sigmoid		1.0981	2.1723	58.5961	51.0672	1.8865	2.3195
Tanh		1.8727	1.1917	54.1711	49.2532	0.97	1.5029
Swish		0.1174	0.1267	146.3747	48.899	0.1195	0.1208
Mish		0.1305	0.1154	56.7093	49.0221	0.1201	0.1216

Fig.9 - California result 2

Rest / Dense 3(25)	ReLu	LeakyRelu	Sigmoid	Tanh	Swish	Mish
ReLu	0.1223	0.1171	2.8244	0.9555	0.1129	0.1229
LeakyRelu	0.1257	0.121	1.6544	0.8188	0.1165	0.1268
Sigmoid	146.3747	55.0924	55.5534	53.6824	146.3747	146.3747
Tanh	50.9006	50.5377	51.3619	49.2532	49.3079	49.1352
Swish	0.1165	0.118	2.0953	1.6458	0.1311	0.132
Mish	0.131	0.1234	2.1753	1.3491	0.1237	0.1216

Fig.10 - California result 3

The numbers in the cell denote the Mean Squared Logarithmic Loss(MSLE).

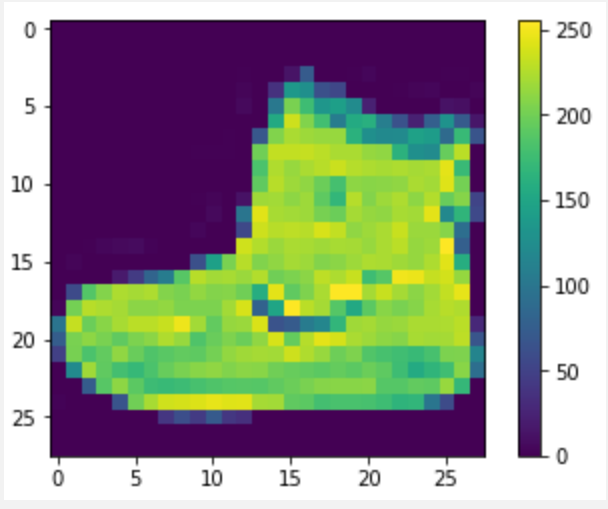
Fashion MNIST:

Fashion-MNIST is a dataset comprising of 28×28 grayscale images of 70,000 fashion products from 10 categories, with 7,000 images per category. The training set has 60,000 images and the test set has 10,000 images. Fashion-MNIST shares the same image size, data format and the structure of training and testing splits with the original MNIST.

Loading data

```
fashion_mnist=keras.datasets.fashion_mnist(train_images,train_labels),(test_images,test_labels)=fashion_mnist.load_data()
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'
print("train_images:", train_images.shape)
print("test_images:", test_images.shape)
train_images: (60000, 28, 28)
train_images: (10000, 28, 28)
```



Train the model

```
model_3.fit(train_images, train_labels, epochs=5, validation_split=0.2)
```

```
# Model a simple 6-layer neural network
model_6 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 11s 236us/step - loss: 0.6
117 - acc: 0.7745 - val_loss: 0.5546 - val_acc: 0.8093
Epoch 2/5
48000/48000 [==============================] - 10s 216us/step - loss: 0.4
404 - acc: 0.8433 - val_loss: 0.4343 - val_acc: 0.8474
Epoch 3/5
48000/48000 [==============================] - 11s 230us/step - loss: 0.3
945 - acc: 0.8593 - val_loss: 0.4056 - val_acc: 0.8628
Epoch 4/5
48000/48000 [==============================] - 10s 212us/step - loss: 0.3
701 - acc: 0.8670 - val_loss: 0.3746 - val_acc: 0.8668
Epoch 5/5
48000/48000 [==============================] - 10s 211us/step - loss: 0.3
499 - acc: 0.8753 - val_loss: 0.3671 - val_acc: 0.8711
10000/10000 [==============================] - 1s 63us/step
Model - 12 layers - test loss: 39.30484714031219
Model - 12 layers - test accuracy: 86.21
```

Resnet:

ResNet50 is a variant of Resnet Model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating points operations. It is a widely used ResNet model and we have explored ResNet50 architecture in depth.

Resnet50 Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```
x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = AveragePooling2D(pool_size = 8)(x)
    y = Flatten()(x)
    outputs = Dense(num_classes,
```

```
                    activation ='softmax',
                    kernel_initializer ='he_normal')(y)

    # Instantiate model.
    model = Model(inputs = inputs, outputs = outputs)
    return model

model.fit_generator(datagen.flow(x_train,y_train,batch_size     =
batch_size),validation_data  =(x_test,  y_test),epochs  =  epochs,
verbose = 1, workers = 4,callbacks = callbacks)
# Score trained model.
scores = model.evaluate(x_test, y_test, verbose = 1)
print('Test accuracy:', scores[1])#Accuracy given in below table
```

Result

Model	Dataset	Relu (Accuracy)	Selu (Accuracy)	Tanh (Accuracy)	Swish (Accuracy)
Resnet50	CIFAR-10	95.84	92.55	92.24	91.60
Resnet50	CIFAR-100	75.72	73.45	64.12	74.46
Resnet50	MNIST	99.56	96.66	99.48	99.53

Model	Dataset	Mean	Standard Deviation	Root Mean Square	Variance
Resnet50	CIFAR-10	0.943	+/- 0.056	0.935	0.025
Resnet50	CIFAR-100	0.787	+/- 0.050	0.789	0.005
Resnet50	MNIST	0.977	+/- 0.019	0.979	0.025

DenseNet:

DenseNet (Dense Convolutional Network) is an architecture that focuses on making the deep learning networks go even deeper, but at the same time making them more efficient to train, by using shorter connections between the layers. DenseNet starts with a basic convolution and pooling layer. Then there is a dense block followed by a transition layer, another dense block followed by a transition layer, another dense block followed by a transition layer, and finally a dense block followed by a classification layer. The first convolution block has 64 filters of size 7x7 and a stride of 2. It is followed by a MaxPooling layer with 3x3 max pooling and a stride of 2.

Result

Model	**Dataset**	**Relu (Accuracy)**	**Selu (Accuracy)**	**Tanh (Accuracy)**	**Swish (Accuracy)**
Densenet	CIFAR-10	83.45	71.00	58.65	72.70
Densenet	CIFAR-100	71.50	70.00	62.48	71.77
Densenet	MNIST	96.33	92.42	78.65	89.70

Model	**Dataset**	**Mean**	**Standard Deviation**	**Root Mean Square**	**Variance**
Densenet	CIFAR-10	0.845	+/- 0.014	0.847	0.031
Densenet	CIFAR-100	0.746	+/- 0.025	0.756	0.010
Densenet	MNIST	0.956	+/- 0.019	0.974	0.005

COMPARISON OF DIFFERENT ACTIVATION FUNCTIONS

As seen from the above experiments and results, ReLU performs best on the classification dataset MNIST while a combination of sigmoid followed by ReLU gives the lowest accuracy with high validation errors. A combination of Swish and ReLU shows results as good as using only ReLU for all the layers.

On regression datasets like California and Boston, the activation functions of sigmoid and tanh exhibit significantly large errors due to the vanishing gradient nature of these functions. In the California dataset results, using Swish in all layers or permutation of Swish with ReLU outperforms other combinations.

For the Boston dataset with activation functions in two consecutive layers- the ordered set of Swish and Tanh gives best results and least errors. For the 3 layer order, activations with Tanh, Mish and ReLU in layer1, layer2 and layer3 respectively work together to provide the lowest error rate. Similarly, the order of ReLU, LeakyReLU and sigmoid is observed to give the worst performance. Detailed results can be found under experiments and results in the previous section.

Recent advances in the study of these functions have led to development of new activation functions like hypersinh, SBAF(Saha-Bora Activation Function) etc. SBAF is defined as-

$$y = \frac{1}{1 + kx^{\alpha}(1-x)^{1-\alpha}}$$

This function when used for training a neural network for habitability classification boasts of an optima and it also doesn't suffer from local oscillation problems.

CONCLUSION

Activation functions suffer from many issues like vanishing gradients, exploding gradients, local oscillation, etc. Better performing activation functions are those which are highly differentiable and vary within limits. Based on the use case, it is beneficial to find optima and different functions give different results based on their nature.

Many different activation functions have been invented that deal with the aforementioned problems and still provide nonlinearity to the neural network model at hand. The functions inside a model need to be ordered in such a way that the effect of one does not get nullified by the next function in line. Fine-tuning parameters like the number of neurons in a hidden layer and using a dropout and learning rate helps achieve the desired results in addition to the activation function selection.

Sigmoid function and its variations usually work best in classification problems. When in doubt, the ReLU function provides better results in most cases and is thus widely used in deep learning networks. While designing your own activation function, it is important to keep in mind that it would be used in the backpropagation of errors and weights hence its effectiveness must be studied on different kinds of datasets.

REFERENCES

1. Parisi, Luca & Ma, Renfei & RaviChandran, Narrendar & Lanzillotta, Matteo. (2021). hyper-sinh: An accurate and reliable function from shallow to deep learning in TensorFlow and Keras. Machine Learning with Applications. 100112. 10.1016/j.mlwa.2021.100112.
2. Saha, Snehanshu & Mathur, Archana & Bora, Kakoli & Agrawal, Surbhi & Basak, Suryoday. (2018). SBAF: A New Activation Function for Artificial Neural Net based Habitability Classification.
3. https://www.v7labs.com/blog/neural-networks-activation-functions
4. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
5. https://iq.opengenus.org/architecture-of-densenet121/
6. Wang Y, Li Y, Song Y, Rong X. The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition. *Applied Sciences*. 2020; 10(5):1897. https://doi.org/10.3390/app10051897