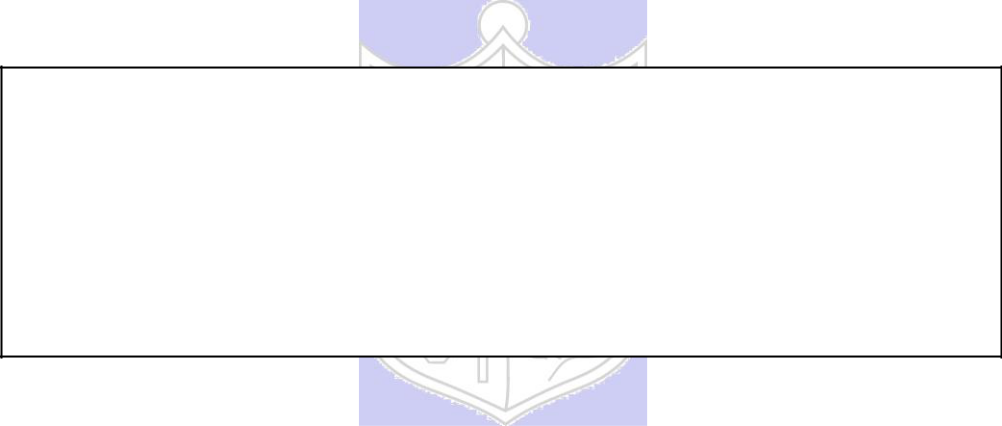
KJSCE/IT/TY BTECH/SEM VI/DL (H)/2023-24



**Experiment No.: 3**

**Title: Deep Neural Network**



(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/TY BTECH/SEM VI/DL (H)/2023-24

**Roll no.: 16010421119 Batch: B4 Experiment No. : 3**

**Aim:** To build deep neural networks capable of learning the complex kinds of relationships

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

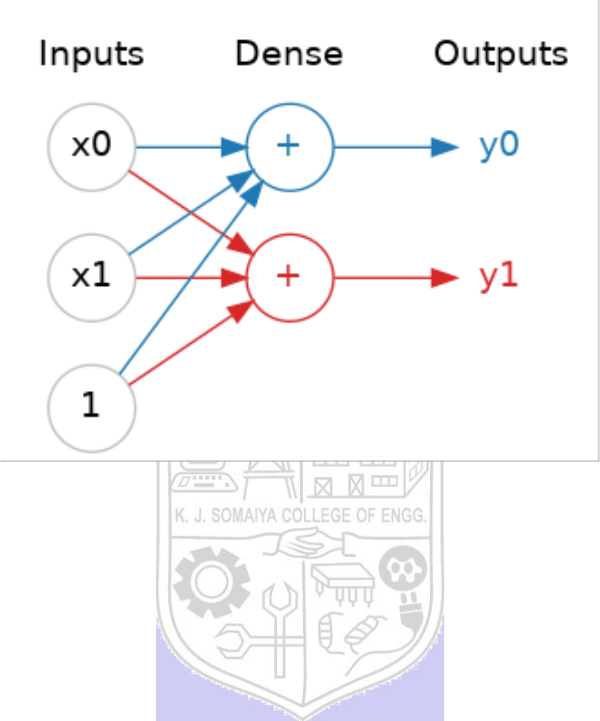
**Resources needed:** Python

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

**Theory:**

**Layers**

Neural networks typically organize their neurons into **layers**. When we collect together linear units having a common set of inputs we get a **dense** layer.



A dense layer of two linear units receiving two inputs and a bias.

You could think of each layer in a neural network as performing some kind of relatively simple transformation. Through a deep stack of layers, a neural network can transform its inputs in more and more complex ways. In a well-trained neural network, each layer is a transformation getting us a little bit closer to a solution.

A "layer" in Keras is a very general kind of thing. A layer can be, essentially, any kind of *data transformation*. Many layers, like the [convolutional](about:blank) and [recurrent](about:blank) layers, transform data through use of neurons and differ primarily in the pattern of connections they form. Others though are used for [feature engineering](about:blank) or just [simple arithmetic.](about:blank)

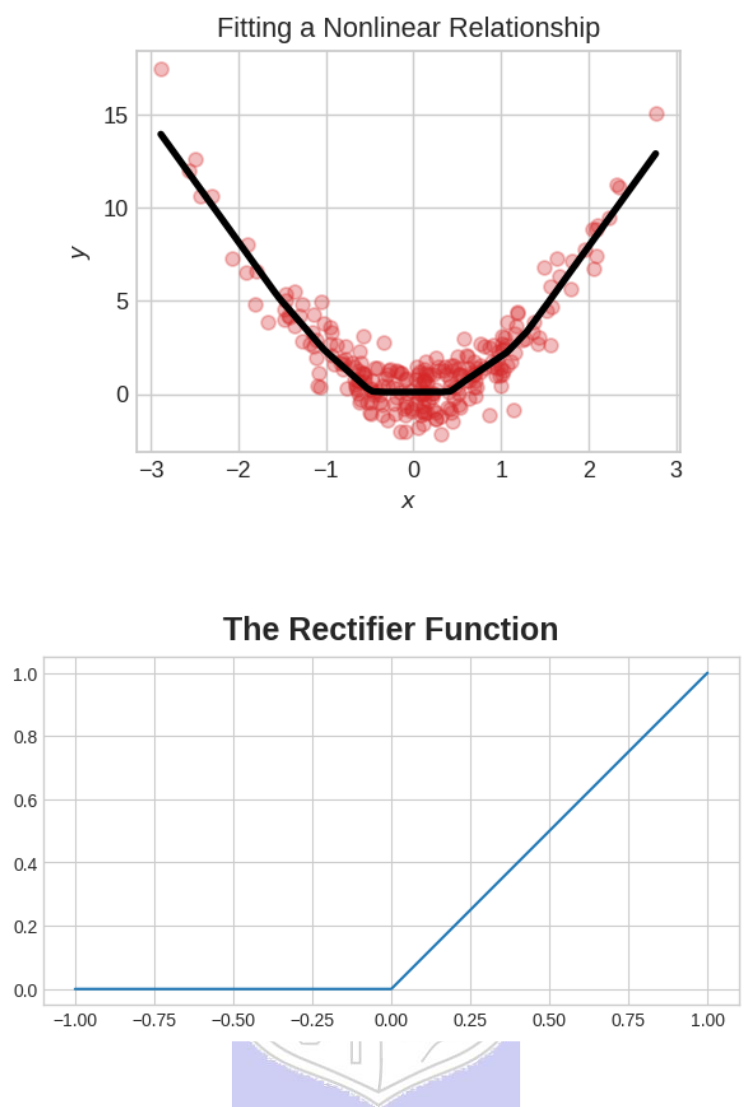
**The Activation Function**

It turns out, however, that two dense layers with nothing in between are no better than a single dense layer by itself. Dense layers by themselves can never move us out of the world of lines and planes. What we need is something nonlinear. What we need are activation functions.

Without activation functions, neural networks can only learn linear relationships. In order to fit curves, we'll need to use activation functions. An activation function is simply some function we apply to each of a layer's outputs (its activations). The most common is the rectifier function max(0,x) .

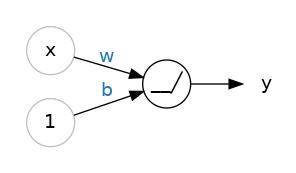
(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/TY BTECH/SEM VI/DL (H)/2023-24



The rectifier function has a graph that's a line with the negative part "rectified" to zero. Applying the function to the outputs of a neuron will put a bend in the data, moving us away from simple lines.

When we attach the rectifier to a linear unit, we get a rectified linear unit or ReLU. (For this reason, it's common to call the rectifier function the "ReLU function".) Applying a ReLU activation to a linear unit means the output becomes max(0, w \* x + b), which we might draw in a diagram like:



A rectified linear unit.

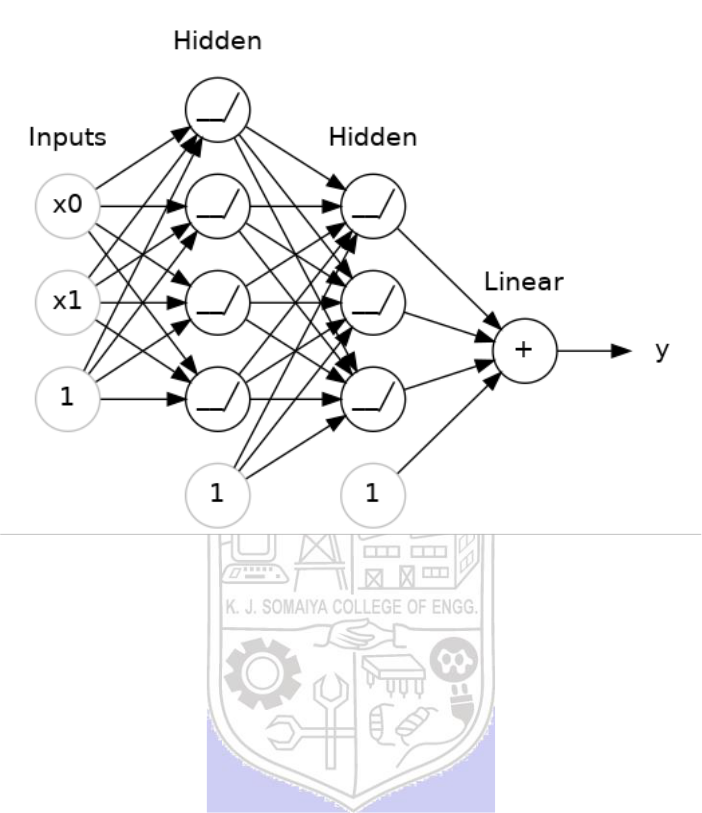
(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/TY BTECH/SEM VI/DL (H)/2023-24

Diagram of a single ReLU. Like a linear unit, but instead of a '+' symbol we now have a hinge '\_/'.

**Stacking Dense Layers**

Now that we have some nonlinearity, let's see how we can stack layers to get complex data transformations.



A stack of dense layers makes a "fully-connected" network.

The layers before the output layer are sometimes called hidden since we never see their outputs directly.

Now, notice that the final (output) layer is a linear unit (meaning, no activation function). That makes this network appropriate to a regression task, where we are trying to predict some arbitrary numeric value. Other tasks (like classification) might require an activation function on the output.

**Building Sequential Models**

The Sequential model we've been using will connect together a list of layers in order from first to last: the first layer gets the input, the last layer produces the output. This creates the model in the figure above:



**Activity:**

1. Download the required dataset.
2. Define the Input Shape.
3. Create a Model with no of hidden layer and output layer.
4. Decide the values of hyperparameters.
5. Try to use different activation functions.
6. Analyze the effect of various values of hyperparameters.
7. Print the developed model.

(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/TY BTECH/SEM VI/DL (H)/2023-24

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Program**

from keras.datasets import mnist

from keras.utils import to\_categorical

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train.reshape((x\_train.shape[0], 28, 28, 1)).astype('float32') / 255

x\_test = x\_test.reshape((x\_test.shape[0], 28, 28, 1)).astype('float32') / 255

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

input\_shape = (28, 28, 1)

from keras.models import Sequential

from keras.layers import Flatten, Dense

def create\_model(hidden\_layers, activation\_function):

model = Sequential()

model.add(Flatten(input\_shape=input\_shape))

for \_ in range(hidden\_layers):

model.add(Dense(128, activation=activation\_function))

model.add(Dense(10, activation='softmax')) # Output layer with 10 classes (digits 0-9)

return model

hidden\_layers = 2

activation\_function = 'relu'

learning\_rate = 0.001

batch\_size = 64

epochs = 10

activation\_functions = ['relu', 'sigmoid', 'tanh']

for activation\_function in activation\_functions:

model = create\_model(hidden\_layers, activation\_function)

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=epochs, batch\_size=batch\_size, validation\_data=(x\_test, y\_test), verbose=0)

print(f"\nModel with {hidden\_layers} hidden layers and {activation\_function} activation function:")

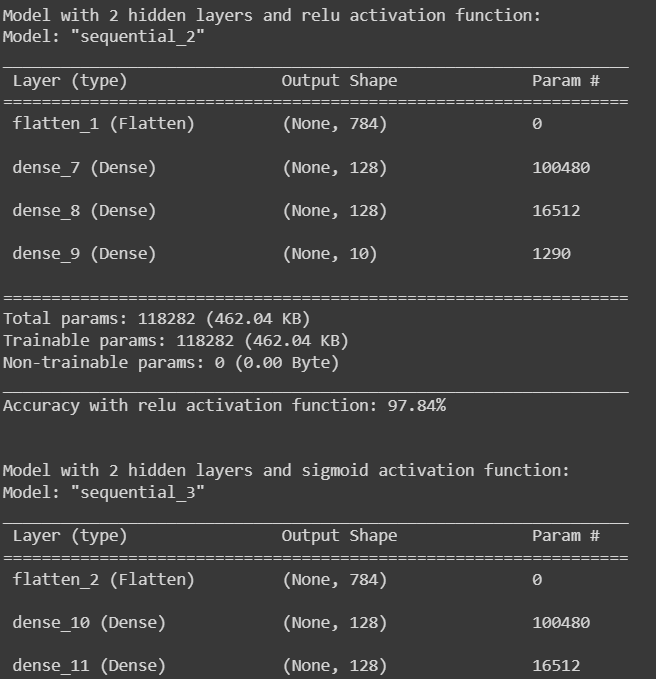
model.summary()

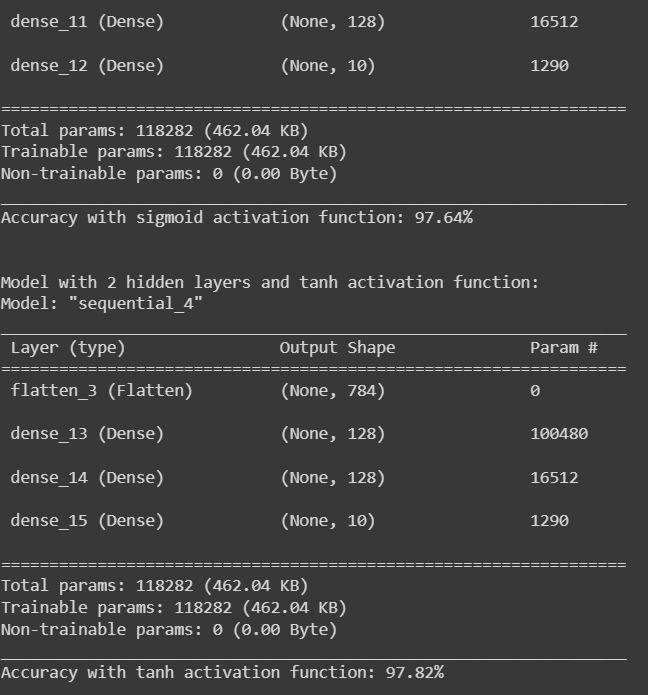
\_, accuracy = model.evaluate(x\_test, y\_test, verbose=0)

print(f"Accuracy with {activation\_function} activation function: {accuracy\*100:.2f}%\n")

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Output:**







**Post lab questions:**

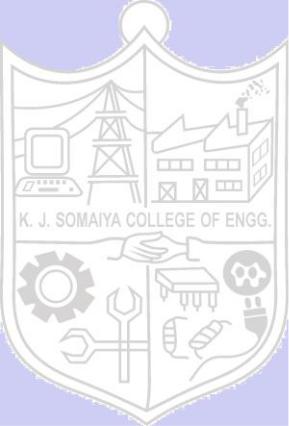
1. Deep learning works well despite of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_problem(s).
   1. Sharp Minima
   2. Numerical instability (vanishing/exploding gradient)
   3. High capacity (susceptible to overfitting)
   4. All of the above
2. The number of neurons in the output layer should match the number of classes (where no of classes are greater than 2) in a supervised learning task. True or False?
   1. True
   2. False
3. List down activation function functions most widely used at hidden layer and output layer.

**Hidden Layer Activation Functions**

1. **ReLU (Rectified Linear Unit)**: This is one of the most commonly used activation functions in deep neural networks because it is simple and reduces the likelihood of the vanishing gradient problem. It outputs the input if it is positive; otherwise, it outputs zero.
2. **Leaky ReLU**: This is a variant of ReLU that allows a small, positive gradient when the unit is not active and the input is less than zero, which helps to keep the gradient flow alive during the training process.
3. **Sigmoid**: This function outputs values between 0 and 1, making it suitable for models where we need to predict the probability as an output. However, it is less commonly used in hidden layers now due to the vanishing gradient problem.
4. **Tanh (Hyperbolic Tangent)**: This function outputs values between -1 and 1. It is similar to the sigmoid but is zero-centered, making it more suitable for layers deep in the network architecture.
5. **ELU (Exponential Linear Unit)**: This function allows for a small gradient when the unit is inactive, which can help to mitigate the vanishing gradient problem. It tends to converge cost to zero faster and produce more accurate results.

**Output Layer Activation Functions**

1. **Softmax**: Used for multi-class classification problems, it outputs a probability distribution across multiple classes, with the sum of the probabilities equal to 1.
2. **Sigmoid**: Used for binary classification problems, it outputs a probability score between 0 and 1, indicating the likelihood of the input belonging to the positive class.
3. **Linear**: Used for regression tasks, it outputs a value directly without applying any activation function, suitable for predicting continuous values.
4. **Softplus**: Similar to ReLU but smoother, it can be used in the output layer for regression problems that require a strictly positive output.



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CO: CO 2 Comprehend the Deep Network concepts.**



**Conclusion: We learnt about deep neural network and implemented it.**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of faculty in-charge with date**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**References:**

**Books/ Journals/ Websites:**

1. Josh Patterson and Adam Gibson, “Deep Learning A Practitioner’s Approach”, O’Reilly Media, 2017
2. Nikhil Buduma, “Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence Algorithms”, O’Reilly Media 2017
3. Ian Goodfellow Yoshua Bengio Aaron Courville. “Deep Learning”, MIT Press 2017

(A Constituent College of Somaiya Vidyavihar University)