Ex No: 4	Training Deep Neural Network with various ways of initialization

AIM:

To train a Deep Neural Network with various ways of initialization.

PROCEDURE:

1. Import the required libraries and packages.
2. Load the MNIST digit image dataset from Keras.
3. Upgrade Keras and install np_utils if necessary.
4. Build the neural network model.
5. Flatten the images to convert them into one-dimensional arrays.
6. Split the data into training and testing sets.
7. Normalize the pixel values of the images to facilitate training.
8. Apply one-hot encoding to the class labels using Keras's utilities.
9. Build a linear stack of layers with the sequential model.
10. Use various weight initializers such as:
    - GlorotUniform
    - HeUniform
    - GlorotNormal
    - Zeros
    - RandomNormal
    - RandomUniform
11. Use Optimizer (optional).
12. Train and test the model.
13. Perform a live prediction with a sample image.

CODE:

```
from keras.datasets import mnist

# loading the dataset

(X_train, y_train), (X_test, y_test) = mnist.load_data()

# let's print the shape of the dataset

print("X_train shape", X_train.shape)

print("y_train shape", y_train.shape)

print("X_test shape", X_test.shape)

print("y_test shape", y_test.shape)

pip install np_utils

!pip install --upgrade keras

# keras imports for the dataset and building our neural network

from keras.datasets import mnist

from keras.models import Sequential
```

```
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D
from keras.utils import to_categorical
# Flattening the images from the 28x28 pixels to 1D 787 pixels
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalizing the data to help with the training
X_train /= 255
X_test /= 255
# one-hot encoding using keras' numpy-related utilities
n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = to_categorical(y_train, n_classes)
Y_test = to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)
import tensorflow as tf
# various ways of weight initialization
from tensorflow.keras import initializers
import tensorflow as tf
# building a linear stack of layers with the sequential
model
model = Sequential()
# hidden layer

# weight initialization – Glorot Uniform
initializer = tf.keras.initializers.GlorotUniform()
model.add(Dense(100, input_shape=(784,),
activation='relu', kernel_initializer=initializer))
# output layer
model.add(Dense(10, activation='softmax'))
# looking at the model summary
model.summary()
# compiling the sequential model
model.compile(loss='categorical_crossentropy',
```

```
metrics=['accuracy'], optimizer='adam')
# training the model for 10 epochs
model.fit(X_train, Y_train, batch_size=128,
epochs=10, validation_data=(X_test, Y_test))
model.save('final_model.keras')
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np
# Load and prepare the image
def load_image(sample_image):
    # Load the image
    img = load_img(sample_image,
color_mode='grayscale', target_size=(28, 28))
    print("Loaded image shape:", img.size)
    # Convert to array
    img = img_to_array(img)
    print("Image array shape after conversion:",
img.shape)
    # Flatten the image array
    img = img.reshape((1, 784))
    # Prepare pixel data
    img = img.astype('float32')
    img = img / 255.0
    return img

# Load an image and predict the class
def run_example(model):  # Pass the model object as
an argument
    # Load the image
    img = load_image('sample_image.jpg')
    predict_value = model.predict(img)
    digit = np.argmax(predict_value)
    print("Predicted digit:", digit)
```

```
# Load the model
model = load_model('final_model.keras')
# Entry point, run the example
run_example(model)  # Pass the model object to the
run_example function
# weight initialization - HeUniform
initializer = tf.keras.initializers.HeUniform()
```

Change this line of code when building the model (shown in previous pages) for different weight initializations

```
# weight initialization – GlorotNormal
initializer = tf.keras.initializers.GlorotNormal()
# weight initialization - Zeros
initializer = tf.keras.initializers.Zeros()
# weight initialization – RandomNormal
initializer = tf.keras.initializers.RandomNormal()
# weight initialization – RandomUniform
initializer = tf.keras.initializers.RandomUniform()
```

OUTPUT:


Predicted Output: **7**

For All initializers except Zero. For Zero, the Predicted Output is 1.

RESULT:

GlorotUniform:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.8063	0.9410	0.6742	0.2089
2	0.9433	0.9548	0.1983	0.1525
3	0.9591	0.9643	0.1428	0.1210
4	0.9689	0.9690	0.1077	0.1082
5	0.9761	0.9706	0.0877	0.0967
6	0.9810	0.9717	0.0701	0.0895
7	0.9819	0.9739	0.0627	0.0824
8	0.9851	0.9763	0.0530	0.0796
9	0.9882	0.9751	0.0428	0.0804
10	0.9898	0.9781	0.0383	0.0725

HeUniform:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.8229	0.9377	0.6399	0.2166
2	0.9447	0.9550	0.1946	0.1523
3	0.9596	0.9640	0.1414	0.1230
4	0.9692	0.9647	0.1102	0.1162
5	0.9742	0.9706	0.0887	0.0970
6	0.9786	0.9734	0.0755	0.0911
7	0.9821	0.9749	0.0620	0.0854
8	0.9845	0.9755	0.0537	0.0811
9	0.9868	0.9771	0.0470	0.0810
10	0.9896	0.9762	0.0377	0.0766

GlorotNormal:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.8113	0.9416	0.6656	0.2007
2	0.9457	0.9580	0.1901	0.1480
3	0.9598	0.9639	0.1374	0.1217
4	0.9693	0.9679	0.1084	0.1036
5	0.9757	0.9720	0.0844	0.0953
6	0.9804	0.9749	0.0712	0.0868
7	0.9830	0.9735	0.0590	0.0834
8	0.9852	0.9747	0.0528	0.0831
9	0.9874	0.9772	0.0454	0.0762
10	0.9896	0.9757	0.0382	0.0793

Zeros:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.1126	0.1135	2.3019	2.3011
2	0.1118	0.1135	2.3015	2.3011
3	0.1106	0.1135	2.3015	2.3010
4	0.1118	0.1135	2.3013	2.3010
5	0.1113	0.1135	2.3016	2.3010
6	0.1101	0.1135	2.3016	2.3010
7	0.1123	0.1135	2.3014	2.3010
8	0.1111	0.1135	2.3013	2.3010
9	0.1154	0.1135	2.3008	2.3011
10	0.1116	0.1135	2.3014	2.3010

RandomNormal:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.1126	0.1135	2.3019	2.3011
2	0.1118	0.1135	2.3015	2.3011
3	0.1106	0.1135	2.3015	2.3010
4	0.1118	0.1135	2.3013	2.3010
5	0.1113	0.1135	2.3016	2.3010
6	0.1101	0.1135	2.3016	2.3010
7	0.1123	0.1135	2.3014	2.3010
8	0.1111	0.1135	2.3013	2.3010
9	0.1154	0.1135	2.3008	2.3011
10	0.1116	0.1135	2.3014	2.3010

RandomUniform:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.8290	0.9343	0.6432	0.2210
2	0.9431	0.9526	0.2014	0.1626
3	0.9583	0.9636	0.1471	0.1307
4	0.9679	0.9672	0.1131	0.1123
5	0.9741	0.9704	0.0919	0.0959
6	0.9781	0.9709	0.0774	0.0952
7	0.9807	0.9727	0.0639	0.0894
8	0.9850	0.9759	0.0513	0.0803
9	0.9868	0.9759	0.0451	0.0748
10	0.9893	0.9741	0.0385	0.0817

INFERENCE:

The initializer “Zeros” has the least accuracy and high loss value in each epoch.

CONCLUSION:

The Deep Neural Network has been trained and tested with various weight initializations for MNIST Digit Dataset and the output has been displayed successfully.