Ex No: 6	Optimization of Deep Neural Network using Mini Batch Gradient Descent and Adam Optimization

AIM:

To implement L2 regularization and Dropout regularization techniques to enhance model accuracy.

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 weight initializers (optional).
11. Use Adam Optimizer.
12. Train and test the model.
13. Perform a live prediction with a sample image.
14. Repeat steps 9 and 10.
15. Use Stochastic Gradient Descent with a batch size of 128 to implement Mini-Batch Gradient Descent.
16. Repeat steps 12 and 13.

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
# building a linear stack of layers with the sequential model
model = Sequential()
# hidden layer
#weight initialization
initializer = tf.keras.initializers.RandomUniform()
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

# Define Mini-Batch Gradient Descent optimizer
optimizer = tf.keras.optimizers.SGD()
# Building a linear stack of layers with the sequential model
model = Sequential()
# Hidden layer
initializer = tf.keras.initializers.RandomUniform()
model.add(Dense(100, input_shape=(784,), activation='relu', kernel_initializer=initializer))
```

```
# Output layer
model.add(Dense(10, activation='softmax'))
# Print the model summary
model.summary()
# Compiling the sequential model with Mini-Batch Gradient Descent optimizer
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=optimizer)
# Training the model for 10 epochs
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_test, Y_test))
# Save the trained model
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)
model = load_model('final_model.keras') # Load the model
# Entry point, run the example
run_example(model)  # Pass the model object to the run_example function
```

OUTPUT:


Predicted Output: **7**

RESULT:

Adam Optimizer:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.8304	0.9426	0.6369	0.2030
2	0.9460	0.9546	0.1927	0.1559
3	0.9616	0.9643	0.1333	0.1223
4	0.9698	0.9689	0.1078	0.1041
5	0.9756	0.9717	0.0883	0.0964
6	0.9798	0.9720	0.0706	0.0906
7	0.9837	0.9748	0.0582	0.0820
8	0.9864	0.9756	0.0505	0.0800
9	0.9885	0.9750	0.0422	0.0780
10	0.9888	0.0388	0.9774	0.0742

Mini Batch Gradient Optimizer:

No of epochs	Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
1	0.5932	0.8645	1.5783	0.6158
2	0.8623	0.8893	0.5812	0.4465
3	0.8850	0.8991	0.4465	0.3869
4	0.8948	0.9035	0.3941	0.3556
5	0.8987	0.9078	0.3663	0.3362
6	0.9038	0.9111	0.3459	0.3204
7	0.9049	0.9139	0.3336	0.3097
8	0.9092	0.9172	0.3225	0.2986
9	0.9124	0.9188	0.3107	0.2909
10	0.9143	0.9196	0.3065	0.2834

CONCLUSION:

L2 regularization and Dropout regularization techniques has been implemented to the dataset to enhance the model accuracy.