Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

Subject/Odd Sem 2023-23/Experiment 2c

Name : Heramb Pawar	**Class/Roll No. :** D16AD/67	**Grade :**

Title of Experiment: Design and Implement a fully connected Deep Neural Network.

objective of Experiment: The goal is to create and execute a deep neural network for classification, aiming to achieve a higher test accuracy by incorporating techniques such as regularization, dropout, and early stopping. This involves designing an optimized neural network and testing it with a suitable dataset.

Outcome of Experiment: The objective is to grasp the process of constructing a neural network using Keras in Python, while also gaining insight into enhancing neural network performance through various techniques such as regularization, dropout, early stopping, and learning rate optimization.

Problem Statement: Create a deep neural network model for a classification task capable of effectively categorizing input data into specific groups. The model will comprise a minimum of two hidden layers and will employ suitable learning algorithms, output functions, and loss functions.

Description / Theory:

Adam Optimizer:

The Adam optimizer, which stands for "Adaptive Moment Estimation," is a widely used optimization algorithm in training neural networks. It combines the advantages of two other well-known optimization methods: RMSProp and momentum-based gradient descent. Adam adjusts the learning rates for individual parameters and keeps track of moving averages for both the gradient and the squared gradient of the loss function. This adaptive characteristic makes it suitable for various deep-learning tasks and helps address some of the limitations seen in other optimization algorithms.

L2 Regularization (Weight Decay):

Incorporating L2 regularization involves adding a penalty term to the loss function for the dense layers of the neural network. This added term is proportional to the square of the weight magnitudes. The purpose is to discourage the model from learning overly intricate features that might lead to overfitting. L2 regularization aids in preventing excessively large weights and encourages the network to concentrate on important features.

Dropout:

Dropout is implemented by adding dropout layers after each dense (hidden) layer. During each training iteration, dropout randomly deactivates a portion of neurons. This regularization method prevents neurons from relying too heavily on specific input features, encouraging the model to learn more robust representations. Dropout effectively reduces overfitting by preventing the network from becoming overly specialized to the training data.

Early Stopping:

During training, the Early Stopping callback is utilized to keep an eye on the validation loss. If the validation loss ceases to improve for a defined number of epochs (known as patience), the training process is halted prematurely. Early stopping is a technique that helps guard against overfitting by preventing the model from memorizing noise present in the training data.

Algorithm/ Pseudo Code / Flowchart (whichever is applicable)

Algorithm: Building and Training a Deep Neural Network for Classification

Input:

- Training dataset (X_train, y_train)
- Validation dataset (X_val, y_val)
- Test dataset (X_test, y_test)
- Number of hidden layers (at least 2)
- Number of neurons in each hidden layer
- Learning algorithm
- Output function
- Loss function
- Hyperparameters (learning rate, batch size, epochs, dropout rate, etc.)

Output:

- Trained neural network model
- Evaluation metrics on the test dataset

```
1. Preprocessing: Normalize or scale the input features if required.

2. Initialize the Neural Network Model:
     Initialize an empty neural network model.
     Add an input layer with the appropriate input shape.

3. Add Hidden Layers:
    For each hidden layer (at least 2):
      Add a fully connected (dense) layer with the specified number of neurons.
      Apply an appropriate activation function (e.g., ReLU) to the layer's output.

4. Add Output Layer:
     Add an output layer with the number of neurons equal to the number of classes.
     Apply the chosen output function (e.g., softmax) to the output layer.
```

5. Compile the Model:
   Compile the model using the chosen learning algorithm, loss function, and relevant metrics.
   Specify the optimizer (e.g., Adam) and learning rate.

6. Train the Model:
   Train the model using the training dataset:
   Use the fit() method, specifying the training data, labels, batch size, and number of epochs.
   Implement early stopping using a callback to prevent overfitting.
   Use dropout layers during training for regularization.

7. Evaluate on Validation Set:
   Evaluate the trained model's performance on the validation dataset:
   Use the evaluate () method to compute validation loss and metrics.
   Track validation accuracy, precision, recall, etc.

8. Hyperparameter Tuning:
   Adjust hyperparameters (e.g., number of neurons, learning rate, dropout rate) based on validation results.

9. Test the Final Model:
   Evaluate the trained model on the test dataset:
   Use the evaluate () method to compute test loss and metrics.
   Calculate test accuracy, precision, recall, etc.

10. Results and Conclusion:
    Summarize the model's performance and the impact of different architectural choices.
    Present evaluation metrics and insights on model generalization.

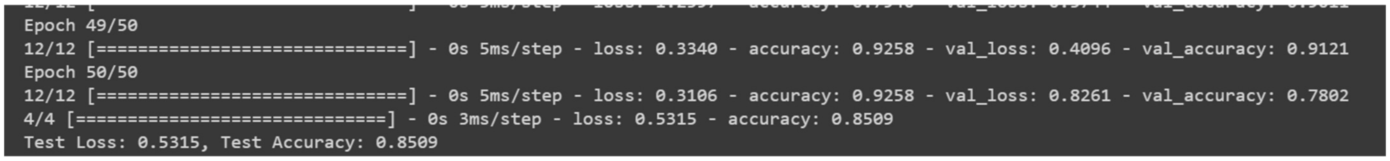
Program :

Before Optimization

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Convert labels to one-hot encoding
y_one_hot = tf.keras.utils.to_categorical(y, num_classes=2)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2,
random_state=42)

# Build the neural network model
model = keras.Sequential([
    layers.Input(shape=(30,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(2, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
# Train the model
batch_size = 32
epochs = 50
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

Output:



```
Epoch 49/50
12/12 [==============================] - 0s 5ms/step - loss: 0.3340 - accuracy: 0.9258 - val_loss: 0.4096 - val_accuracy: 0.9121
Epoch 50/50
12/12 [==============================] - 0s 5ms/step - loss: 0.3106 - accuracy: 0.9258 - val_loss: 0.8261 - val_accuracy: 0.7802
4/4 [==============================] - 0s 3ms/step - loss: 0.5315 - accuracy: 0.8509
Test Loss: 0.5315, Test Accuracy: 0.8509
```

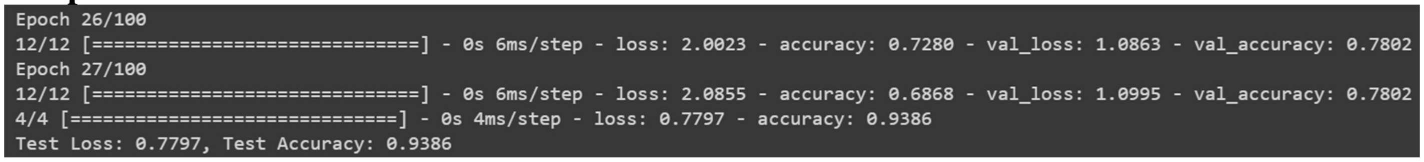
Accuracy is 85%

After Optimization

```
# Build the neural network model with regularization, dropout, and early stopping
model = keras.Sequential([
   layers.Input(shape=(30,)),
   layers.Dense(128, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.01)),
   layers.Dropout(0.5),  # Adding dropout with a rate of 0.5
   layers.Dense(64, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.01)),
   layers.Dropout(0.5),  # Adding dropout with a rate of 0.5
   layers.Dense(2, activation='softmax')
])
```

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Implement early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the model with early stopping
batch_size = 32
epochs = 100
model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.2, callbacks=[early_stopping])
```

Output:



```
Epoch 26/100
12/12 [==============================] - 0s 6ms/step - loss: 2.0023 - accuracy: 0.7280 - val_loss: 1.0863 - val_accuracy: 0.7802
Epoch 27/100
12/12 [==============================] - 0s 6ms/step - loss: 2.0855 - accuracy: 0.6868 - val_loss: 1.0995 - val_accuracy: 0.7802
4/4 [==============================] - 0s 4ms/step - loss: 0.7797 - accuracy: 0.9386
Test Loss: 0.7797, Test Accuracy: 0.9386
```

Accuracy is 93%

Results and Discussions:

The model employed the Adam Optimizer and was trained on a breast cancer dataset for classification. Initially, without optimization techniques, it achieved a test accuracy of 85% after 50 epochs. However, following the incorporation of optimization methods such as L2 Regularization, Dropout, and early stopping, the test accuracy significantly increased to 93%. This experiment provided valuable insights into constructing a deep neural network using Keras and optimizing it with various techniques when working with a dataset.