**Experiment 6b**

**Aim: Design and implement a simple RNN model with tensor/keras and check accuracy**

**Description:**

This program implements an **RNN (Recurrent Neural Network)** model using TensorFlow/Keras to classify handwritten digits from the **MNIST dataset**. The MNIST dataset consists of grayscale images of handwritten digits, each of size 28×28 and the task is to classify these images into 10 classes (0 to 9).

### ****Sequential Nature of Data Representation****

* Although MNIST images are 2D, they can be treated as a sequence by processing one row or one column of pixels at a time.
* In RNN:
  + Each row (or column) of the image can be considered as a time step.
  + The network learns to extract patterns by iterating sequentially through these rows/columns.

**Operations Performed :**

#### ****Loading and Preprocessing Data****

The **MNIST dataset** is loaded using mnist.load\_data() then **Normalization and One-Hot Encoding is applied on the data**

#### ****2. Defining the RNN Model****

**Input Shape**: The input shape is specified as 28×28 where 28 represents the time steps (rows of the image) and another dimension of size 28 corresponds to the features (columns of the image).

**Model Layers**:

* 1. **Input Layer**:
     + Specifies the input shape as (28,28)
  2. **Reshape Layer**:
     + Reshapes the 28x28 image into 28 time steps, each with 28 features, suitable for processing by the RNN.
  3. **LSTM Layer**:
     + A recurrent layer with 128 hidden units (units=hidden\_size).
     + Uses **tanh activation** to process sequential data and capture temporal patterns.
  4. **Dense Layer**:
     + The output layer with 10 neurons (one per class), uses **softmax activation** to output probabilities for classification.

#### ****3. Model Compilation****

* **Loss Function**: categorical\_crossentropy is used for multiclass classification.
* **Optimizer**: Adam optimizer is selected for efficient and adaptive gradient updates.
* **Metrics**: Accuracy is used to evaluate model performance.

#### ****4. Training and Validation****

The model is trained using model.fit:

* **Epochs**: 10 iterations over the dataset.
* **Batch Size**: 128 samples per batch.
* **Validation Data**: The test set is used for validation during training.

#### ****5. Evaluation****

The model is evaluated on the test set to compute:

* **Test Loss**: Measures the prediction error.
* **Test Accuracy**: Proportion of correct predictions.

### ****Key Functions****

1. **tf.keras.layers.LSTM**: Processes sequential data by maintaining hidden and cell states. And captures long-term dependencies using forget, input, and output gates.
2. **to\_categorical:** Converts integer labels into a one-hot vector, making them suitable for the softmax output layer.
3. **Adam **Optimizer**:** An adaptive learning rate optimization algorithm that improves convergence.
4. **categorical\_crossentropy **Loss****: Measures the difference between the predicted probability distribution and the true distribution.

**Result Analysis:**

|  |  |
| --- | --- |
| Optimizer | Adam, RMSprop, SGD |
| Learning rate | 0.001,0.01,0.0001 |
| No of units in LSTM(hidden\_size) | 64,128,256 |
| Batch size | 64,128 |
| No of epochs | 10,20,30 |
| Activation Function | Tanh, relu |
| Dropout Rate | 0.2,0.3,0.5 |
| Loss Function | categorical\_crossentropy, sparse\_categorical\_crossentropy |

The performance of the model is analyzed using different hyperparameters. The table below summarizes the results:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activation Function | Optimizer | Loss Function | Lr=  0.001 | Lr=  0.01 | Lr=  0.0001 | Hidden size =64 | Hidden size =128 | Hidden size =256 | Dropout  0.2 | Dropout  0.3 | Dropout  0.5 | No of Epochs=  10 | No of Epochs=  20 | No of Epochs=  30 |
| Tanh | Adam | categorical\_crossentropy, |  |  |  |  |  |  |  |  |  |  |  |  |
| Tanh | SGD | categorical\_crossentropy, |  |  |  |  |  |  |  |  |  |  |  |  |
| Tanh | RMSprop | categorical\_crossentropy, |  |  |  |  |  |  |  |  |  |  |  |  |
| Relu | Adam | sparse\_categorical\_crossentropy |  |  |  |  |  |  |  |  |  |  |  |  |
| Relu | SGD | sparse\_categorical\_crossentropy |  |  |  |  |  |  |  |  |  |  |  |  |
| Relu | RMSprop | sparse\_categorical\_crossentropy |  |  |  |  |  |  |  |  |  |  |  |  |

**Conclusion**

* Activation Function:
* Adam Optimizer: The Adam optimizer with a learning rate of \_\_\_\_\_\_ performed the \_\_\_\_\_\_,
* SGD Optimizer: The SGD optimizer performed \_\_\_\_\_\_than Adam across all learning rates, indicating that Adam is \_\_ suited for this specific problem.
* Learning Rate: Higher learning rates (\_\_\_\_) resulted in \_\_ performance for both optimizers compared to lower learning rates (0.001 and 0.01).
* Loss Function: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_