Experiment No 4

Aim :-To implement the Stochastic Gradient Descent (SGD) optimization algorithm using TensorFlow.

Apparatus :-
Programming language: Python 3.x
Libraries/Frameworks: TensorFlow 2.x, NumPy, Matplotlib
Google Colab

Theory :-
What is Stochastic Gradient Descent (SGD)
Stochastic Gradient Descent (SGD) is an optimization technique used to minimize the loss function in machine learning models, especially neural networks. It is a variant of gradient descent that updates model parameters using the gradient computed from a single training example at each iteration, instead of the entire dataset.

Mathematically, if the model parameters are Θ, the SGD update rule at iteration t with learning rate η and a randomly selected training sample

$for\ i\ in\ range\ (m):$

$$\theta_j = \theta_j - \alpha(\widehat{y^i} - y^i)X^i_j$$

where J is the loss function.

Why does SGD show fluctuations in loss
Since SGD updates parameters after seeing only one (or a few) data points, the computed gradient can be noisy or less accurate compared to the full dataset gradient. This causes:

- Fluctuations or noise in the loss curve: Loss values might jump up or down from one iteration to the next instead of decreasing smoothly.
- Faster exploration of parameter space: The noise can sometimes help SGD escape local minima or saddle points that a deterministic full-batch gradient descent might get stuck in.

These fluctuations are intrinsic to the stochastic nature of the algorithm.

Mini-Batch Gradient Descent vs SGD
Mini-Batch Gradient Descent computes the gradient using a small batch of b samples (e.g., 32 or 64) instead of a single sample (SGD) or the entire dataset (batch gradient descent).

Advantages of Mini-Batch over Pure SGD:
Reduced Variance: Using a batch smooths the noise in gradient estimates, resulting in more stable updates and less fluctuation in loss.
Efficient Computation: Modern hardware (like GPUs) is optimized for parallel computations on batches, improving speed.
Better Convergence: Balances noise and computational efficiency to converge faster and more reliably.

When is Pure SGD Used
For very large datasets or streaming data where only one sample can be processed at a time.
When memory constraints prevent batching.
Sometimes in early training stages to explore parameter space widely.

Is SGD better than Mini-Batch Gradient Descent
SGD (batch size = 1) has high noise which might cause unstable training but sometimes helps avoid local minima.
Mini-batch gradient descent is typically preferred in practice due to its balance between noise reduction and efficiency.
Batch size choice matters: Small batches (like 32 or 64) offer good stochasticity; very large batches resemble full gradient descent, which can be slow and get stuck.

Procedure:-
1. Data Preparation:
Generated synthetic data with 1000 samples and 20 features each.
Binary target labels (0 or 1) were randomly assigned.

2. Model Design:
Constructed a simple feedforward neural network with one hidden layer (64 neurons, ReLU activation) and an output layer with sigmoid activation for binary classification.

3. Compilation:
Compiled the model using the SGD optimizer with a learning rate of 0.01.
Used binary cross-entropy as the loss function.
Monitored accuracy as a metric.

4. Training:
Trained the model for 10 epochs with a batch size of 32.
Recorded the loss after every batch using a custom Keras callback.

5. Visualization:
Plotted the mini-batch loss values to observe convergence behavior during training.

Code

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt

# Sample data
X_train = np.random.rand(1000, 20)
y_train = np.random.randint(0, 2, size=(1000, 1))

# Define model
model = models.Sequential([
        layers.Dense(64, activation='relu', input_shape=(20,)),
        layers.Dense(1, activation='sigmoid')
])

# Compile model with SGD optimizer
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
        loss='binary_crossentropy',
        metrics=['accuracy'])

# Callback to record loss per batch
class BatchLossHistory(tf.keras.callbacks.Callback):
        def on_train_batch_end(self, batch, logs=None):
        self.losses.append(logs['loss'])

        def on_epoch_begin(self, epoch, logs=None):
        self.losses = []

batch_loss_history = BatchLossHistory()

# Train model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, callbacks=[batch_loss_history],
verbose=1)

# Plot batch losses
plt.plot(batch_loss_history.losses)
plt.xlabel('Batch')
plt.ylabel('Loss')
plt.title('Mini-Batch Loss per Batch (SGD)')
plt.show()
```

Observations:-

The loss per mini-batch generally decreased over time, demonstrating that the model was learning and converging.

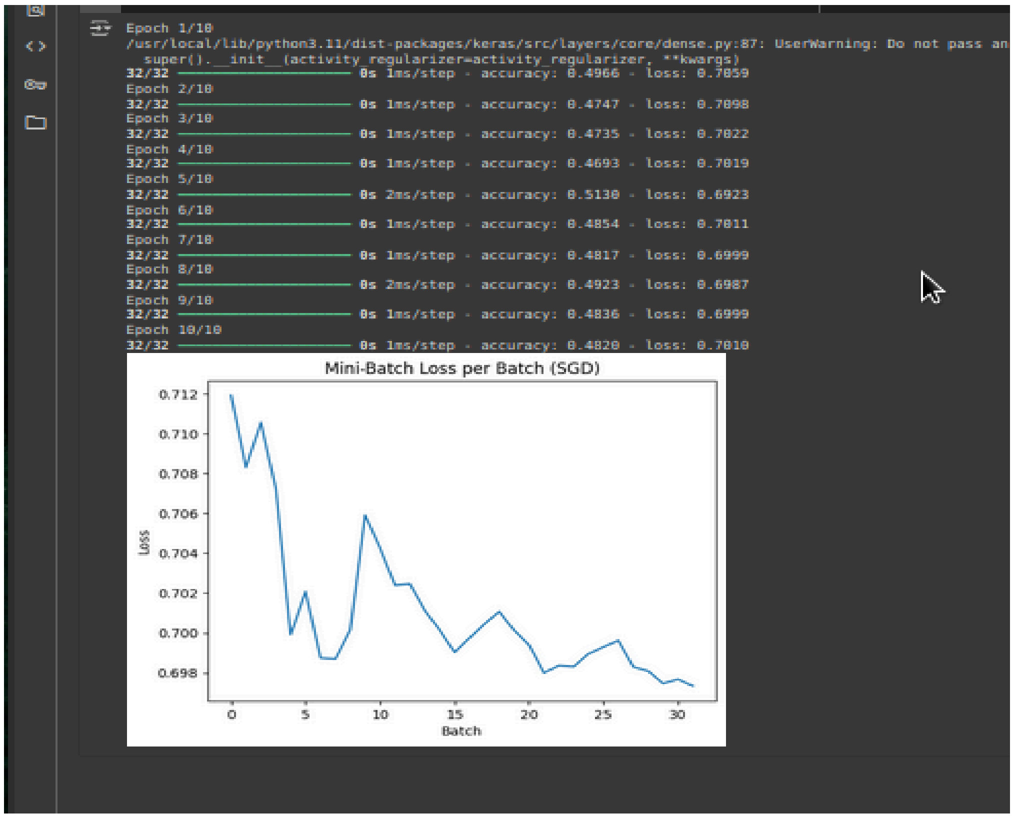
SGD with mini-batches effectively optimized the neural network parameters.

Accuracy improved steadily, indicating good learning despite the randomness in data.

Conclusion:-

The practical successfully demonstrated the implementation of Stochastic Gradient Descent using TensorFlow. Mini-batch loss visualization provided insight into the optimization process and confirmed that SGD can efficiently train models by updating parameters based on small batches of data. This approach is widely used in training deep learning models due to its balance between computational efficiency and convergence stability.

Result:-



```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4966 - loss: 0.7059
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4747 - loss: 0.7098
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4735 - loss: 0.7022
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4693 - loss: 0.7019
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5130 - loss: 0.6923
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4854 - loss: 0.7011
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4817 - loss: 0.6999
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4923 - loss: 0.6987
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4836 - loss: 0.6999
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4820 - loss: 0.7010
```