Lab Design 1 – Generating Synthetic Data and Binary Classification

Name – **Ravi Tarun Prasad Nimmalapudi**

Course - **CPSC 5960 03 PyTorch Lab Design**

❖ Problem Statement:

The primary objective of this lab is to classify geometric shapes based on randomly generated data points. Considering this to be the basic classification problem, classification is one of the most fundamental tasks in machine learning where the goal is to assign a category label to a given input data. Creating artificial datasets that depict various forms, including triangles, stars, squares, and circles, is the challenge at hand. Every data point is associated with a point that is part of one of these forms and has (x, y) coordinates.

The objective is to categorize the input points into the appropriate shape categories by designing, training, and assessing a neural network model based on PyTorch. This lab would show how well deep learning models excel in non-linear decision-making tasks, such as recognizing shapes in noisy, randomly generated data.

❖ Approach:

The approach for this shape classification task is straightforward and aligns with standard steps for any simple supervised learning problem. The process involves generating synthetic data for various geometric shapes, preprocessing the data, building a PyTorch-based model, and training it for classification. The focus is understanding the significance of creating synthetic datasets, the ability of neural networks to recognize complex patterns, and how PyTorch streamlines the entire process.

Before diving into the workflow, let us briefly discuss the significance of synthetic data generation, the role of neural networks in classification, and the reason why PyTorch is the best framework for this purpose.

Why generate synthetic data?

Synthetic data generation plays a key role in machine learning and deep learning tasks considering the main challenges to be when real data is scarce, unavailable, or too expensive to collect.

Few of the scenarios highlighting the importance of synthetic data are:

1. Availability of real-world data:
   In many cases, real-world data might not be available or could be hard to collect due to privacy concerns, regulatory issues etc.

2. Cost and time consuming:
   Collecting and labelling real-world data is definitely time consuming and costly. Whereas synthetic data can be generated quickly at a very low cost as we did for this task.

3. Creating balance:
   Considering real-world datasets there is a lot of noise and imbalance. Generating synthetic data allows oversampling, helping to balance the dataset and improving model performance.

Why PyTorch is the best framework?

In the context of classification problems and synthetic data generation, PyTorch is an excellent choice because it offers powerful tools for both creating synthetic datasets and building efficient models to classify them. In order to train machine learning models, PyTorch makes it simple to generate a variety of shapes (such as squares, triangles, circles, etc.) with randomized attributes like size and position as per our problem. Because of PyTorch's versatility, we can create distinct data creation pipelines that instantly generate sizable datasets, guaranteeing that our models are trained on a variety of representative samples.

❖ **Workflow:**

▪ **Understanding the data and the problem:**

The first step would be to understand the nature of the problem, which is a simple binary classification problem which involves classifying different shapes generated synthetically. This is similar to any other machine learning task where understanding the data is the first and most crucial step.

Coming to the data, in simple terms the input data will be the attributes of the shapes. These attributes are nothing but numerical values representing the geometric properties of the shapes which would form the input features that the model will use for classification.

For Example:

- A circle would have attributes like radius and position.
- A square would have side length, position and orientation etc.

- **Data Generation:**

In this lab, one of our main focusing areas is generating synthetic data. So, we would be generating synthetic data for two distinct shapes namely circles and squares for now. The task would be creating data points that represent each of these shapes and assign them corresponding labels for classification and then using this synthetically generated data to train and evaluate a binary classifier.

The data generation process is crucial because real-world datasets may not always be available or suitable for specific tasks, especially in scenarios where we want to evaluate models on tasks like shape classification. By creating synthetic data, we may test machine learning models in a controlled setting without requiring manually labeled data. Additionally, it makes it simple to modify data properties (such as size, noise, and class distribution) in order to assess classification algorithm's performance more accurately.

1. <u>Circle Shape Data Generation:</u>

   We begin by choosing angles between 0 and 2π (360 degrees) at random in order to create circular data. These angles represent the direction of the points around the circle. Next, we add a small amount of noise to a set radius number to produce random radii for the spots. This simulates faults found in the real world by ensuring that the points are not precisely on the edge of the circle. Using these angles and radii, we calculate the x and y coordinates of each point using basic trigonometric formulas. The points generated this way are then labeled as "Class 0," representing the circle shape.

2. <u>Square Shape Data Generation:</u>

   For generating the square data, within a square region, we generate x and y co-ordinates at random for the square data. The coordinates are chosen such that they fall within a square with a defined side length. We introduce random noise into the x and y values to give the points a more realistic appearance, causing them to deviate slightly from the square's perimeter. The points created in this manner are labeled as "Class 1," indicating the square shape.

3. <u>Combining the generated data:</u>

   Now as we have the square and circle data generated, we combine them into one dataset. The x and y coordinates of both sets of points are stacked together to accomplish this. This creates the feature matrix, in which the coordinates of each point are represented by a row.

The labels for each point are also combined into a single array, where the label for circle points is 0 and the label for square points is 1. This combined dataset is now ready for use in training a classifier.

4. Splitting the data:

   Now in order to assess the performance of our classification model, we divide the combined data into two parts as per any classic machine learning classification problem, one part for training and the other part for testing.
   The test set is used to evaluate the model's ability to forecast fresh, unseen data, while the training set is used to educate the model how to classify the points. So as to make sure that the model is tested on new data after it has been trained, we usually use a 70-30 split, in which 70% of the data is utilized for training and 30% for testing.

- **Model Initialization and Design:**

For this task, we use PyTorch to create a simple classification neural network designed to classify geometric shapes based on the synthetic data we generated. The core of the classification process is the neural network, which predicts the relevant shape class (such as "circle," "square," or other shapes if added) based on the two-dimensional coordinates of the points.

Using PyTorch's torch.nn module, we can efficiently define and customize the architecture of our neural network. This module offers a number of pre-built elements that simplify and modularize model design, including fully connected layers (for changing input features), activation functions (for adding non-linearity), and other helpful tools.

Brief view of the model:

1. Input Layer:
   Takes the 2D co-ordinate data points as input.

2. Hidden Layers:
   These process the features using linear transformations and apply activation functions to capture/learn the complex patterns from the data.

3. Output Layer:
   This layer simply maps the transformed features to the output classes.

The flexibility of PyTorch allows us to expand this model as needed. For example, additional hidden layers or units can be added to enhance its capacity, or we can experiment with different activation functions to improve performance.

- **Model Training and Evaluation:**

Once the neural network is defined, the next step is training it using the synthetic data we generated. During training, the model learns to minimize the difference between its predictions and the actual labels by adjusting its internal parameters. This procedure takes advantage of PyTorch's robust tools, including loss functions like cross-entropy loss and optimization algorithms like Adam and Stochastic Gradient Descent (SGD), which are designed for classification jobs. The training procedure is smooth because of PyTorch's automatic differentiation feature (autograd), which effectively calculates the gradients required for backpropagation.

After training, we evaluate the model's performance on a separate set which we split initially. To do this, unknown data must be fed into the model, its predictions must be compared to the actual labels, and measures like accuracy must be computed to measure its effectiveness. It is simple to carry out these assessments and gain understanding of the behavior of the model thanks to PyTorch's user-friendly framework.

- **Future Scope / Problem Extension:**

This lab highlights the powerful capabilities of PyTorch in addressing fundamental machine learning tasks, specifically synthetic data generation and classification. We illustrated how to create an efficient classification model and simulate data for controlled tests by utilizing PyTorch's versatility and extensive toolkit. This provides a fundamental method for both academic assignments and real-world applications when labeled data may be hard to find or unavailable.

However, the scope of this work can be extended in few more directions:

- Introducing more shapes:

  Apart from simple shapes like circles and squares, more complex shapes like triangles, ellipses and pentagons or introduce irregular patterns challenging the model to handle more features and improve its adaptability.

- Multiclass Classification:

  The current setup focuses on binary classification, but it can be extended to a multiclass problem where the model predicts several shape categories. This would offer a more thorough evaluation of the model's performance.

- Performance Optimization:

  Evaluating more complex training methods like transfer learning, hyperparameter tuning, and data augmentation can improve the model's accuracy and efficiency.