TO RUN THE CODE PLEASE MAKE SURE IN COLAB, YOU HAVE THE FOLLOWING DATA PATH- "/content/data/cora/"

Documentation: Graph Convolutional Network (GCN) for Node Classification

Overview

This documentation explains the implementation of a Graph Convolutional Network (GCN) using PyTorch for node classification on graph-structured data. The implementation covers data loading, GCN model definition, training, and evaluation.

Requirements

Python 3.x

PyTorch (including torch.nn, torch.optim)

NumPy

SciPy

scikit-learn (for evaluation metrics)

Implementation Details

Data Loading

The load_data function is used to load the graph data. It processes the data from a specified dataset (e.g., Cora dataset) into features, labels, and adjacency matrix. The adjacency matrix is normalized, and features are also normalized.

Model Definition

Two main classes are defined for the GCN model:

GraphConvolution

This class defines a single graph convolutional layer.

This class defines the overall GCN model, which consists of two GraphConvolution layers (one hidden layer and one output layer) and uses dropout for regularization.

Training the Model

The model is trained using the following steps:

Initialize the GCN model with the number of features, hidden units, classes, and dropout rate.

Define the optimizer (e.g., Adam).

Transfer the model and data to the appropriate device (CPU or GPU).

Run the training loop, which includes:

Forward pass of the model.

Computing the loss (e.g., negative log-likelihood loss).

Backpropagation and optimization.

Validation step for monitoring the performance.

Testing the Model

After training, the model is evaluated on a test set. The evaluation metrics include accuracy, precision, recall, and F1 score.

Functions

train: Function to perform a single epoch of training.

test: Function to evaluate the model on the test set.

accuracy: Utility function to calculate accuracy.

Other utility functions include encode_onehot, feature_normalize, adj_normalize, etc.

Usage

Load the data: adj, features, labels, idx_train, idx_val, idx_test = load_data()

Define the model: model = GCN(...)

Train the model: for epoch in range(num_epochs): train(...)

Evaluate the model: test(model, ...)

Error Handling

Ensure all tensors and the model are on the same device (CPU/GPU).

Make sure to handle the shapes and types of tensors correctly.

Catch and handle exceptions related to file handling and data loading.

Conclusion

This GCN implementation provides a basic framework for node classification tasks on graph-structured data. It can be extended and modified for more complex graph-based machine learning tasks.