GNN tutorial for a chemical classification task using the BBBP dataset

Alex Dong

**Abstract**

Graph neural networks (GNNs) have seen rapid development in recent years and exhibit large popularity in the field of drug discovery with many studies having shown that GNNs yield more promising results than traditional descriptor-based models. Despite its potential, GNNs can be relatively complex models to implement. This project presents a tutorial with an end-to-end implementation of a GNN for working with molecular data using the PyTorch Geometric library along with training and testing of the GNN model on the Blood Brain Barrier Penetration (BBBP) benchmark dataset.

**Introduction**

The unique properties of the blood vessels which vascularize the central nervous system, designated as the blood-brain barrier (BBB), tightly regulate the movement of ions, molecules, and cells between the blood and the brain [1]. Precise regulation of the BBB provides proper neuronal function and protects neural tissue from toxins and pathogens; however, this same regulation provides a challenge for the discovery and administration of drugs designed to target the brain. Quick and reliable prediction of blood-brain barrier penetration (BBBP) is therefore important for drug development [1,2]. The BBBP dataset consists of binary labels for over 2000 compounds based on their permeability properties and has been adopted by MoleculeNet as a benchmark dataset for machine learning.

A wide variety of machine learning methods have been applied in property prediction and tested on the BBBP dataset. Developments in deep learning have revolutionized this task thanks to the capacity of deep learning models to learn intricate relationships between structures and their properties. Broadly speaking, deep learning models can be classified into two types: traditional descriptor-based models and graph-based models. In descriptor-based models, molecular descriptors and/or fingerprints are used as input, then a specific deep learning architecture is employed to train the model. In graph-based models, the basic chemical information of each molecule encoded in a graph data structure is used as input, then a graph-based deep learning algorithm, such as GNN, is used to train the model.

**Materials and methods**

**The BBBP dataset**

The performance of the GNN developed in this tutorial is assessed on the BBBP dataset which is one of the benchmark datasets adopted by MoleculeNet for machine learning classification tasks. This dataset contains the molecular names, SMILES strings, and the binary labels of 2050 molecules based on their ability to permeate the BBB. Issues with this dataset are that the ratio of items in each class is extremely unbalanced with a ratio of roughly 3 to 1 and the overall number of datapoints is small.

After importing the data file, canonical SMILES were generated from the SMILES strings in the dataset as a method of detecting any invalid molecules. Using the RDKit package, SMILES strings from the data file were converted into RDKit molecule objects and then back into canonical SMILES strings. Molecules which could not be successfully processed by RDKit were excluded from the dataset, and following this preprocessing, the number of instances in the dataset came down to 1975.

**Molecular graph representation**

Molecules are represented as an undirected graph. The nodes are represented by a node label matrix where each row corresponds to the one-hot encoding of a node’s associated atom features. The edges are represented by an edge label matrix where each row is a one-hot encoding of the associated edge’s bond features. The atomic features encoded into the node label matrix were atom type, number heavy neighbors, formal charge, hybridization type, whether the atom is in a ring, whether the atom is aromatic, atomic mass, van der Waal radius, and covalent radius. The bond features encoded into the edge label matrix were bond type, whether the bond is conjugated, and whether the bond is in a ring. These features were calculated using RDKit and one-hot encoded using self-defined functions. This encoding might be overly comprehensive as it includes all available features that can be computed by RDKit.

The matrices for each molecule and their corresponding labels are used to generate PyTorch Geometric Data objects which are flexible wrappers for graph structured data. The graphs and labels are split into training, validation, and test sets with a ratio of 8:1:1. Stratified split is used since the dataset is biased and so that training and evaluation is more consistent.

**Graph neural network (GNN)**

Here the PyTorch Geometric library is used to implement a GNN. The split datasets are each wrapped in Torch Geometric DataLoaders with a batch size of 64 chosen arbitrarily. A basic Network model is then implemented which can be summarized as follows:

1. The model performs two graph convolutions.
2. Then the node embeddings are pooled.
3. The pooled embeddings are run through two fully connected layers.
4. The model outputs a class membership probability using a sigmoid function.

Since the bond features of a molecule are relevant to the prediction task, a graph convolution which utilizes edge features was implemented into the defined neural network model. For this task, the torch\_geometric.nn.NNConv layer was implemented. The convolution in this layer can be written as:

Where is a neural network parameterized by a set of weights , and is a weight matrix for node labels. The fact that the neural network is parameterized based on edge labels allows the weights to vary for different edge labels.

A pooling layer was added to deal with varying graph sizes. For this task, the global\_add\_pool layer from torch\_geometric.nn was implemented. This layer adds all the node embeddings of a graph into a fixed-size output so that the data can be run through subsequent fully connected layers. The sigmoid function then outputs class probabilities for this classification task.

**Results and discussion**

The model was trained using binary cross entropy as a loss function. After each epoch, the for the validation set was calculated to track model performance. Training the model over 20 epochs, it is evident that there is slight overfitting of the model on the training set as we consistently see lower loss on the training set. Analyzing the performance, we see that on average, the on the validation set are close to 1, indicating high precision and recall. There is a trend where the increase for a couple of epochs and then start to decrease again. Taking this into account, the model with the best was saved and evaluated on the test set. The on the test set was computed to be and the ROC-AUC score on the test set was computed to be 0.75.

**Conclusion**

This tutorial showed the end-to-end implementation of a GNN for molecular data with evaluation on the BBBP classification task. First, molecular data from the dataset was validated using RDKit. Next, atom and bond features were computed and one-hot encoded into node and edge matrices. This data was wrapped in the PyTorch geometric object so that it could be fed into the GNN. Stratified splitting was performed on the data with 8/1/1 splits for the training, validation, and test sets. A basic GNN model was defined with two convolutional layers, a global pooling layer, and two fully connected layers. Training and evaluation are like that of other neural network models. For the BBBP classification task, performance was measured using and ROC-AUC.

Using ROC-AUC, we may compare our results to the model and the results reported in Table 4 of Jiang’s paper [3], however, we will see that the basic model developed in this tutorial is nothing special. There are several factors should be considered for further development of this model. The number of atom and bond features one-hot encoded in the graph was probably more thorough than the task required. Further research may give indication as to which features are the most important and which features might be extraneous. The elimination of extraneous features may also improve the efficiency of the model by decreasing the size of the graph matrices. In addition, different architectures may change the performance of the model. Factors in the architecture that might vary performance are the number of convolutional layers, the type of convolutional layer, the type of pooling layer, the number and type of fully connected layers, along with batch size, optimizers, and loss functions. When testing these variations, it may be helpful to set aside a smaller initial training set to make the process more efficient. Finally, since this is a classification task, it may also be helpful to look at different thresholds and evaluation metrics. For example, the favors similar precision and recall, however, for the task of predicting BBBP it may be more important to minimize type I error and prioritize precision. All these factors should be considered to improve the base model implemented in this tutorial.

**References**

1. Daneman R, Prat A. The blood-brain barrier. Cold Spring Harb Perspect Biol. 2015 Jan 5;7(1):a020412. doi: 10.1101/cshperspect.a020412. PMID: 25561720; PMCID: PMC4292164.
2. Sakiyama H, Fukuda M, Okuno T. Prediction of Blood-Brain Barrier Penetration (BBBP) Based on Molecular Descriptors of the Free-Form and In-Blood-Form Datasets. Molecules. 2021 Dec 7;26(24):7428. doi: 10.3390/molecules26247428. PMID: 34946509; PMCID: PMC8708321.
3. Jiang, D., Wu, Z., Hsieh, CY. *et al.* Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models. *J Cheminform* **13**, 12 (2021). <https://doi.org/10.1186/s13321-020-00479-8>