Graph Convolution on BACE regression

Niccolai Lorenzo, Pozzoli Davide

Project 3 report for
Deep Learning

Artificial Intelligence
University of Bologna
Italy
31/05/21

Contents

GitHub: https://github.com/treyvian/Molecular-Property-Prediction

1 Introduction

The project we chose is a Deep Learning application to Molecular Property Prediction, in concise terms the exploitation of AI to predict a value related to a given molecule. This kind of challenge can have impact on the discovery of new drugs, since the selection of molecules with useful properties is a complex process. Stanford University's MoleculeNet is a benchmark containing multiple datasets related to quantum mechanics, biophysics, physiology and physical chemistry [3]. We worked on the BACE dataset, which is under the Biophysics section.

1.1 BACE Dataset

Beta-Secretase 1 (BACE) is a transmembrane aspartic-acid protease human protein encoded by the BACE gene. BACE is essential for the generation of beta-amyloid peptide in neural tissue, a component of amyloid plaques widely believed to be critical in the development of Alzheimer's, rendering BACE an attractive therapeutic target for this devastating disease.

	Class	pIC50	MW	AlogP	HBA	HBD	RB	HeavyAtomCount	ChiralCenterCount
count	1513.000000	1513.000000	1513.000000	1513.000000	1513.000000	1513.000000	1513.000000	1513.000000	1513.000000
mean	0.456709	6.521991	479.661988	3.177080	3.732981	2.001322	8.049570	34.089227	0.522802
std	0.498287	1.342417	122.083053	1.396633	1.444778	1.629343	4.741135	8.520088	1.162539
min	0.000000	2.544546	138.187000	-4.361100	0.000000	0.000000	0.000000	10.000000	0.000000
25%	0.000000	5.585027	389.331300	2.335500	3.000000	0.000000	4.000000	28.000000	0.000000
50%	0.000000	6.761954	463.628300	3.171300	4.000000	2.000000	7.000000	33.000000	0.000000
75%	1.000000	7.540000	564.639530	4.015500	4.000000	3.000000	11.000000	40.000000	1.000000
max	1.000000	10.522879	1350.473300	7.617400	12.000000	15.000000	40.000000	97.000000	10.000000

Figure 1: The dataset description

The dataset is formed by 1513 compounds and their target values, so it is not so big in size. This is due to the difficulty of obtaining data of new molecules. The molecule ('mol' field in the dataset) is represented with the SMILES format, which is a standardized codification to express molecules as strings, some work has already been done trying to train a neural network directly with this format [2]. We chose a different path, there are multiple techniques to translate a SMILES string in order to encode information, the

more natural way to represent a molecule is a graph: the nodes are the atoms and the edges are the bonds. This process is also called featurization or creating a fingerprint for the molecule, we will deepen this further in the report.

1.2 Task

The BACE dataset comprehends 2 tasks:

- Classification: Class field
- Regressions: IC50 field

while on the classification task good results have been obtained even with traditional machine learning techniques, the regression of IC50 is more challenging and a Deep Learning approach could be the right way to pursue this task.

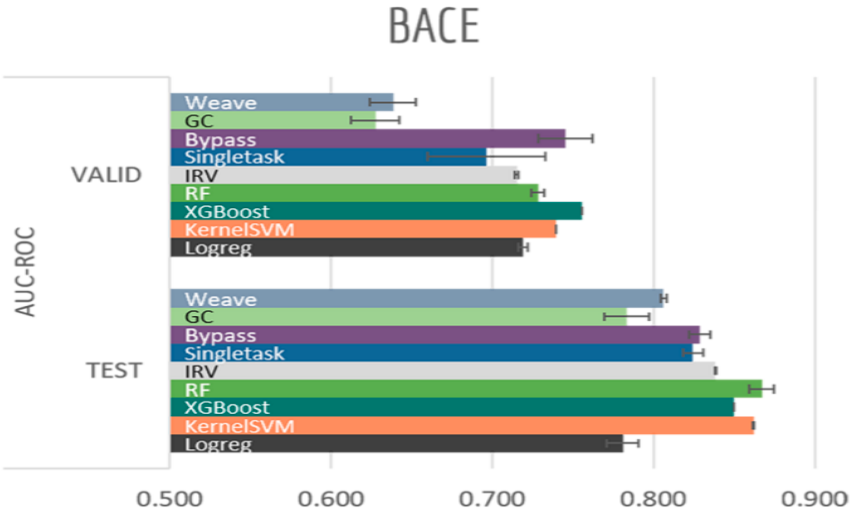
1.2.1 Metrics Used

The main metrics used to evaluate regression tasks are:

- **Pearson R2 score**: is the proportion of the variance in the dependent variable that is predictable from the independent variable
- **RMSE**: It is the Root of the Mean Squared Error, which measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

1.3 State of the Art

The MoleculeNet paper reports some results on the classification task - which are reported in the image right below -, while for the regression a baseline result using graph convolution has been stated by Mufei Li. Anyway there is not so much literature about the latter, so we did not have a lot of guidelines to evaluate our results.



2 Libraries used

We used Keras to build the network, but also other libraries for data featurization and useful built-in functions. Initially we considered Rdkit - as advised in the slides - to obtain the molecule graph from the SMILES string, then we looked at Deepchem and saw that it includes Rdkit functions and augment them to automatize and simplify the workflow.

2.1 RDKit

RDKit is a collection of cheminformatics and machine-learning software written in C++ and Python. Even though we are not using RDKit directly, it stands at the base of the class Featurizer in DeepChem.

2.2 Deepchem

DeepChem maintains an extensive collection of utilities to enable scientific deep learning including classes for loading scientific datasets, processing them, transforming them, splitting them up, and learning from them. Behind the scenes DeepChem uses a variety of other machine learning frameworks such as scikit-learn, TensorFlow, and XGBoost.

2.2.1 Splitters

In DeepChem, a method of splitting samples into multiple datasets is defined by a Splitter object. Choosing an appropriate method for your data is very important. Otherwise, your trained model may seem to work much better than it really does.

Consider a typical drug development pipeline. You might begin by screening many thousands of molecules to see if they bind to your target of interest. Once you find one that seems to work, you try to optimize it by testing thousands of minor variations on it, looking for one that binds more strongly. Then perhaps you test it in animals and find it has unacceptable toxicity, so you try more variations to fix the problems.

This has an important consequence for chemical datasets: they often include lots of molecules that are very similar to each other. If you split the data into training and test sets in a naive way, the training set will include many molecules that are very similar to the ones in the test set, even if they are not exactly identical. As a result, the model may do very well on the test set, but then fail badly when you try to use it on other data that is less similar to the training data.

- **RandomSplitter** This is one of the simplest splitters. It just selects samples for the training, validation, and test sets in a completely random way.
- **ScaffoldSplitter** This splitter identifies the scaffold that forms the core of each molecule, and ensures that all molecules with the same scaffold are put into the same dataset.
- **ButinaSplitter** This is another splitter that tries to address the problem of similar molecules. It clusters them based on their molecular fingerprints, so that ones with similar fingerprints will tend to be in the same dataset.

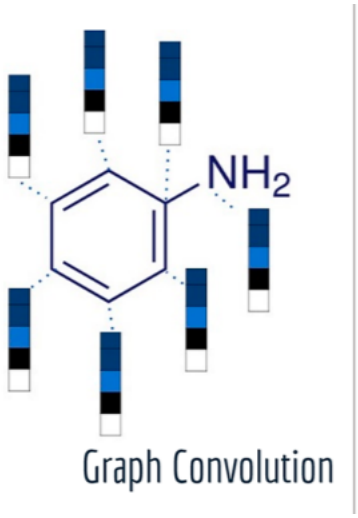
2.2.2 Featurizer

A "Featurizer" is chunk of code which transforms raw input data into a processed form suitable for deep learning. In particular we employed the ConvMolFeaturizer, this class perform the featurization to implement graph convolutions. Using Rdkit this class maps a SMILES string into a Graph of atoms and bonds with certain features obtained from the molecule. This featurizer uses the representations based on Duvenaud graph convolutions [1], which construct a vector of descriptors for each atom in a molecule. The featurizer computes that vector of local descriptors. The featurizer has the following setting parameters:

- master_atom: if true create a fake atom with bonds to every other atom. the initialization is the mean of the other atom features in the molecule.
- use_chirality: if true then make the resulting atom features aware of the chirality of the molecules in question
- atom_properties: properties in the RDKit Mol object to use as additional atom-level features in the larger molecular feature. If None, then no atom-level properties are used.
- per_atom_fragmentation: If True, then multiple "atom-depleted" versions of each molecule will be created (using featurize() method). For each

molecule, atoms are removed one at a time and the resulting molecule is featurized.

We used only use_chirality, since the performances with the other options were not improving.



3 Data Transformation

3.1 Dataset Loading

To load the original BACE dataset we used the DeepChem method dc.molnet.load_bace_regression, which takes as arguments the featurizer and the splitter. This method processes the dataset through the featurizer, building a graph object starting from the SMILES string in the 'mol' field, then the splitter splits the dataset in train, validation and loss with 80/10/10 proportions. The splitted data format is made of two main parts:

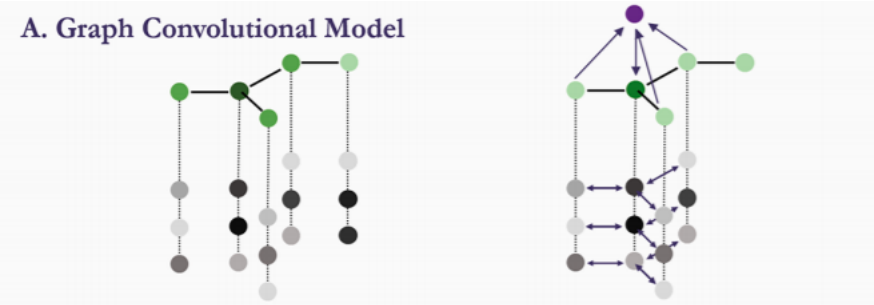
- **X**: field which contains a Graph object built by the featurizer as stated in 2.2.2
- **y**: field which contains our target IC50 value

3.2 Data Generation

After the loading we have the splitted data made by graphs(ConvMol objects), to feed it into the network we need to obtain tensors. To perform this mapping we wrote the data_generator function that given a batch of data generates the lists of inputs and labels, whose values are Numpy arrays. It uses the ConvMol.agglomerate_mols method from DeepChem to transform a molecule graph in a set of arrays to encode atom features, index a flattened version of the atoms, the membership of atoms in molecules and the adjacency of an atom.

4 Model

The technique applied by this network is graph convolution, which is schematized in the image below. The model is described inside the class *MyGraphConvModel* which extends tensorflow.keras.Model. The class is composed by the constructor, in which all the layer of the model are defined and the method call, in which the links between layers are defined.



For the model we used as a starting point the model presented in the 6^{th} file of the of DeepChem repository on GitHub. Since it was created to fulfill a classification on another dataset we had to modify it.

4.1 Layers

```
Model: "my_graph_conv_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
graph_conv (GraphConv)       multiple                  212352
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
graph_conv_1 (GraphConv)     multiple                  346752
_________________________________________________________________
graph_gather (GraphGather)   multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  65792
_________________________________________________________________
dense_1 (Dense)              multiple                  32896
_________________________________________________________________
dense_2 (Dense)              multiple                  8256
_________________________________________________________________
dense_3 (Dense)              multiple                  2080
_________________________________________________________________
dense_4 (Dense)              multiple                  33
=================================================================
Total params: 668,161
Trainable params: 668,161
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 2: Keras Model Summary

The model is composed by 9 layers in total, including the input and the output layer.

We first insert the data into two Graph Convolution layers with a Dropout layer between the two. The activation function for the convolutional layers as well as for the dense layers is the ReLU function(Rectified Linear Unit). The second convolution layer takes also in input the combination between the previous output and the initial input. The output of these layers is then passed to a graph gather layer, which conveys the results of the convolutions into a single tensor.

The output tensor is then carried out through a 4 Dense layer to extract the value we want to predict, each layers has half the size of the previous one, ending with a dense layer of a single node which represents our regression output.

4.1.1 Graph Convolution

This layer implements the graph convolution introduced by Duvenaud in his paper[1]. The graph convolution combines per-node feature vectors in a non-

linear fashion with the feature vectors for neighboring nodes, as the inputs give atom features and graph topology encoded in vectors from the data_generator function. This layer sum each atom features with his neighbors', then all the values are combined with the layer's weights and added to the final output. This "blends" information in local neighborhoods of a graph. Such process is a step forward from classic circular fingerprints - such as ECFP - because it replaces every operation with a differentiable analog.

4.1.2 Dropout

The Dropout layer randomly sets input units to 0 with a frequency of chosen rate at each step during training, which helps to prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

4.1.3 Graph Gather

Many graph convolutional networks manipulate feature vectors per graph-node. For a molecule for example, each node might represent an atom, and the network would manipulate atomic feature vectors that summarize the local chemistry of the atom. However, at the end of the application, we will likely want to work with a molecule level feature representation. This layer creates a graph level feature vector by combining all the node-level feature vectors.

4.1.4 Dense

Dense is the basic feed forward layer, it implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).

4.2 Hyper-parameters

The hyperparameters of the model have been tested and perfected along the project. The most important one though is the Batch size, which normally influences the quality of the network convergence in relation to the training speed. In this case GraphGather depends on the Batch size. The GraphConv layer pool all nodes from all graphs in a batch that's being processed. The GraphGather reasembles these jumbled node feature vectors into per-graph feature vectors. We tried 32,64 and 128 because usually the powers of 2 are the mose used sizes.

4.3 Experiment Settings

In section 2.2 we reported Deepchem's options for splitters and featurizers, we ran basic tests with few samples for training to choose both: this is fully reported in section 5. For the splitter initially we thought that Scaffold could

be a viable choice to make the splits uniform, but the better results came with the Random Splitter, so a quite standard approach. The featurizer on the other hand is strictly linked to the model structure, so the choice was forced due to the Graph Convolution model. With the random splitter comes also a greater variation in the model results based on how data is splitted.

We also tried to change the dropout layer rate from 0.2 to 0.5 in order to decrease the amount of overfitting and have a better accuracy on the test set. The result was not what we expected since both the test and the training accuracy were a bit lower than the previous results.

5 Results

In Table 1 we report our results on 100 epochs with various splitters and batch sizes, from those we chose to employ Random Splitter and 64 Batch Size.

	32	64	128
Random	88/65/0.23	88/69/0.20	84/53/0.34
Scaffold	74/41/1.12	69/22/0.89	74/59/0.95
Butina	74/44/0.48	79/44/0.35	74/53/0.48

Table 1: The rows correspond to different Splitters, the columns correspond to different Batch sizes. The matrix values are Training R2 score/Test R2 score/Test MSE.

With these parameters we tried to push further the model's performances by incrementing the epochs (the training time). Of course we monitored the loss to stop when the decreasing became too small. Our best results came with 512 epochs:

- **Train R2**: 0.985 ± 0.007
- **Test R2**: 0.715 ± 0.055
- **Test RMSE**: 0.577 ± 0.2

Through the tests we did we used also the model proposed by Deepchem *dc.models.GraphConvModel(1, mode='regression')*. It allowed us to monitor how well, or how bad our model did, almost guiding us in the right direction. We trained both models with the same number of epochs and the same dataset to have comparable results. The accuracy obtained on the training set were almost the same especially with the number of epochs set to 512. Some differences can be seen in the test set where our model perform a little better almost every time.

6 Conclusion

With our work we tried to predict an important indicator for biophysics research on the BACE dataset, given the limited hardware power and the dataset size the results are quite good. Even if they could be surely improved with a bigger model and with more data.

The results we obtained are good but not perfect we never got past 80 R2 score on the test split, so there is more room to grow and improve the model. For example to further prevent overfitting early stopping with validation run could be introduced. Another interesting technique that can be applied on this challenge is Message Passing, which could also gather information from bonds.

References

[1] David Duvenaud et al. "Convolutional networks on graphs for learning molecular fingerprints". English (US). In: *Advances in Neural Information Processing Systems* 2015-January (2015). Funding Information: We thank Edward Pyzer-Knapp, Jennifer Wei, and Samsung Advanced Institute of Technology for their support. This work was partially funded by NSF IIS-1421780.; 29th Annual Conference on Neural Information Processing Systems, NIPS 2015 ; Conference date: 07-12-2015 Through 12-12-2015, pp. 2224–2232. ISSN: 1049-5258.

[2] Hirohara M., Saito Y., and Koda Y. "Convolutional neural network based on SMILES representation of compounds for detecting chemical motif". In: (2018).

[3] Zhenqin Wu et al. "MoleculeNet: A Benchmark for Molecular Machine Learning". In: (2018). arXiv: 1703.00564 [cs.LG].