# Molecular Property Prediction

*Deep Learning - project 3*

Ciaramicoli Michele

Urbinati Cristian

a.a. 2020/2021

# Contents

# List of Figures

# Abstract

$\beta$-secretase 1 inhibitor drugs are some promising medication for Alzheimer's disease. This project has the aim to develop a neural network that can predict the IC50 value (half maximal inhibitory concentration) of some enzymes of interest and also provide a qualitative label describing weather or not the molecule is good for Alzheimer prevention. The dataset suitable for this application is called BACE-1 and is provided by Stanford University. The molecules are treated as graphs, we first extract information from their atoms and bonds and then, we set up a recurrent network based on message passing framework: a *message* is exchanged between atoms to update their hidden state; then bond states are updated starting from the state of the associated atoms and subsequently a global state is used to combine all these states and to propagate the learned features. The global state is the one used at the end to obtain the predictions.

# 1 | Problem description

Alzheimer's disease (AD) research has recently shifted his focus to a new and promising drug: the $\beta$-secretase inhibitor. Much of AD research has investigated the amyloid cascade hypothesis, which postulates that AD is caused by changes in amyloid beta (A$\beta$) stability and aggregation. Blocking A$\beta$ production by inhibiting the first protease required for its generation, $\beta$-secretase 1, may be the next step in blocking AD progression [1].
The aim on this work is to provide a quantitative prediction of the value of the human $\beta$-secretase 1's IC50 and also a qualitative prediction, so a binary label, indicating if the molecule is or not a good candidate for Alzheimer prevention.

## 1.1 What are we predicting

The quantitative value IC50 stays for "half maximal inhibitory concentration" and it's a measure in molar concentration of how much a particular substance is needed to inhibit, in vitro, a given biological process or biological component by 50%. The biological components, in this case, are a set of inhibitors of human $\beta$-secretase 1 and we will predict in particular its logarithmic conversion pIC50. For what concerns the qualitative prediction instead, this is a parameter strongly related to the IC50 value and it will indicates if the enzyme is suitable or not for $\beta$-secretase 1 inhibition; we will obtain also this parameter as a neural network output.

## 1.2 BACE-1 dataset

A dataset of these inhibitors of human $\beta$-secretase 1 is available at MoleculeNet.ai (by Stanford University) called the BACE-1 dataset. It provides the composition of the molecule described as strings in the Simplified Molecular-Input Line-Entry System (SMILES) format, the binary label of the class, the value of IC50 and a suggested division of data for train, validation and test along with a series of molecule property.
In parallel with the dataset we used the RDKit library. This was an useful tool to convert back the SMILES into a graph representation or an image of the molecule. For example, starting from a string like:

$$O1CC[C@@H](NC(=O)[C@@H](Cc2cc3cc(ccc3nc2N)-c2ccccc2C)C)CC1(C)C$$

we can get the representation in fig.1.1

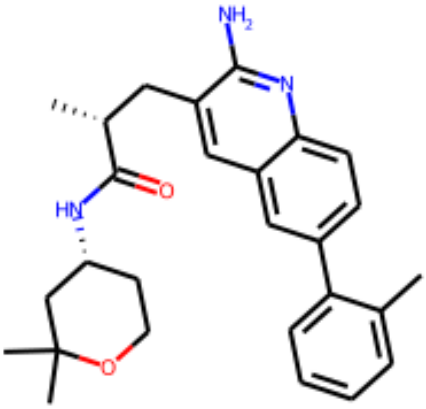


Figure 1.1: Representation of a molecule extracted using RDKit library

Through this library is also possible to extract the global feature of the molecule such as its weight, the number of heavy atoms, the number of ring that it presents and many other, or even generate a fingerprint of 0s and 1s indicating if certain common patterns are present or not.
There are also functions to extract features from single atoms and bonds of the molecule and these are the ones we have decided to use to generate the data to feed our model.

# 2 | Spatial graph approaches

We denote a graph as $G = (V, E)$ where $|V| = N$ is the number of nodes in the graph and $|E| = N^e$ is the number of edges. $A \in R^{N \times N}$ is the adjacency matrix.

For graph learning tasks, we can distinguish two types of classification:

- Node-level tasks focus on nodes, which include node classification, node regression, etc. Node classification tries to categorize nodes into several classes, and node regression predicts a continuous value for each node.
- Edge-level tasks are edge classification and link prediction, which require the model to classify edge types or predict whether there is a link existing between two given nodes.

To build a model that works with graphs let's first categorize computational modules of our interest:

- **Propagation Module**. The propagation modules includes convolutional operators and recurrent operators and are used to propagate information between nodes so that the aggregated information could capture both feature and topological information.
- **Pooling Module**. When we need the representations of high-level subgraphs or graphs, pooling modules are needed to extract information from nodes.

### Propagation

Convolution operators are modules created to generalize convolutions on pictures to the graph domain; we are interested in what is called the spatial

approach that defines convolutions directly on the graph based on its topology. Recurrent operators have the only difference with convolution operators in the fact that layers in convolution operators use different weights while layers in recurrent operators share same weights. They are composed by gate mechanism like GRU or LSTM and in the propagation step they improve the long-term propagation of information across the graph structure. They run a fixed number of training steps without the guarantee of convergence.
The node first aggregates information from its neighbors. Then the GRU-like update function incorporates information from the other nodes and from the previous timestep to update each node's hidden state (last illustration in fig. 2.1).

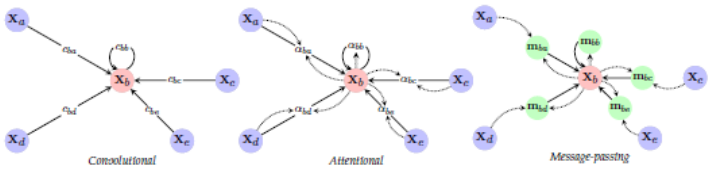


Figure 2.1: Different types of propagation approaches

### Pooling

Pooling modules are needed because as in computer vision, a convolutional layer is usually followed by a pooling layer to get more general features. Complicated and large-scale graphs usually carry rich hierarchical structures which are of great importance for node-level and graph-level classification tasks. [2]

## 2.1 MPNN

Let's analyze in particular the framework we used in out project called Message Passing Neural Network. We will use the spatial approach that define convolutions directly on the graph based on the graph topology as said. This approach was defined by Gilmer et al. [3] to operate on undirected graphs $G$ with node features $x_v$ and edge features $e_{vw}$. The forward pass has two phases: a message passing phase and a readout phase.

The **message passing phase** runs for M time steps and is defined in terms of message functions $M_t$ and vertex update functions $U_t$ . During the message passing phase the hidden state $h_v^t$ at each node in the graph are updated based on messages $m_v^{t+1}$according to:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, h_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

where in the sum, $N(v)$ denotes the neighbors of $v$ in graph $G$.

The **readout phase** computes a feature vector for the whole graph applying a readout function $R$ on hidden states according to:

$$\hat{y} = R(\{h_v^M | v \in G\})$$

The message functions $M_t$ , vertex update functions $U_t$ , and readout function $R$ are all learned differentiable functions. $R$ operates on the set of node states and must be invariant to permutations of the node in order for the MPNN to be invariant to graph isomorphism.

Note one could also learn edge features in an MPNN by introducing hidden states for all edges in the graph $h_{e_{vw}}^t$ and updating them using an update function that takes into account the nodes associated to that edge:

$$h_{e_{vw}}^{t+1} = U_t'(h_{e_{vw}}^t, h_v^t, h_w^t)$$

# 3 | Proposed model

We have decided to use for our proposal a library called nfp (neural fingerprint) used also in the study "Message-passing neural networks for high-throughput polymer screening" [4] in order to perform the message passing and the regression from a graph-based features representation to a vector-based one.

Inputs for the neural network are generated from the molecules' SMILES strings, and consists of discrete node types, edge types, and the connectivity matrix.

Atoms are categorized into discrete types based on their atomic symbol, degree of bonding, and whether or not they are present in an aromatic ring. Bonds are similarly categorized into discrete types based on the atoms involved, their type (single, double, triple, or aromatic), conjugation and presence in a ring.

The message passing step was implemented using the matrix multiplication method, where messages $m$ are passed between neighboring atoms:

$$m_v^{t+1} = \sum_{w \in N(v)} A_{e_{vw}} h_w^t$$

where $A_{e_{vw}}$ is a learned weight matrix for each bond type.

The update step was implemented as a Gate Recurrent Unit:

$$h_v^{t+1} = GRU(h_v^t, m_v^{t+1})$$

Initial atom and bond embeddings, $h_v^0$ are initialized at zero and learned as additional model parameters. The dimension of features encoded in the embedding layer was chosen to be 128, followed by $M = 3$ message-passing phases.

In the libraries we operated, Peter et al. [4] use a readout function that only takes account of the final hidden state of the recurrent atom unit to generate a whole-graph feature vector $\hat{y}$:

$$\hat{y} = \sum_{v \in G} \sigma(W, h_v^M)$$

where $W$ is a learned matrix. The dimension of $\hat{y}$ was chosen to be 1024. This summed fingerprint is then passed through a series of fully connected layers with batch normalization and ReLU activation functions (dimensions 512, 256 and 8 respectively), before being passed to an output layer corresponding to the prediction we want to perform.

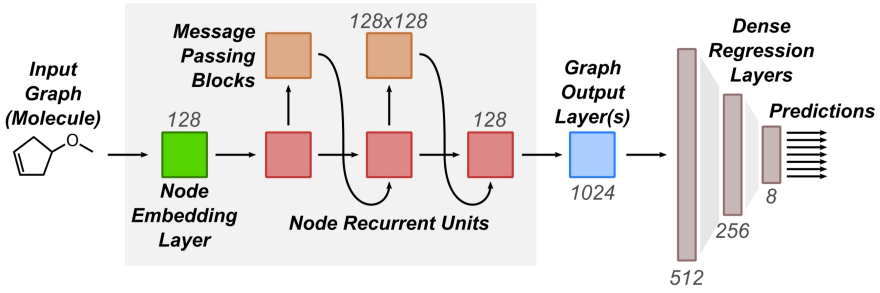


Figure 3.1: A representation of the nfp model

## 3.1 Molecule featurization

One important step has been to define the input for the network in a coherent way. The idea was to extract features respectively for atoms and bonds of the enzyme starting from the SMILES and using the functions made available by RDKit.

In the code we have defined two functions that do exactly that:

- `atom_featurizer`: given the SMILES molecular representation, concatenates into a string various atom features like the atom symbol, atom degree, the formal charge, the number of radical electrons, etc.
- `bond_featuraizer`: given the SMILES molecular representation, concatenates into a string the following informations: the symbols of the

atoms involved in the bond separated by dash, the type (single, double, triple, or aromatic) of the bond, the presence of the conjugation phenomenon and the belonging to a ring.

These two method are then passed as parameter to the function `construct_feature_matrices` already defined in the nfp package that performs the preprocessing of the data through a tokenization from strings describing atoms and bonds to integer numbers, in a way that equal strings are mapped to the same number.
In particular, the result of this function is a dictionary for each molecule modeled as follows:

- `n_atom`: integer representing the number of atoms in the molecule
- `n_bond`: integer representing the number of bonds in the molecule
- `atom`: array of shape (n_atom,) representing the list of atom classes
- `bond`: array of shape (n_bond,) representing the list of bond classes
- `connectivity`: matrix of shape (n_bond, 2) which represents the connections so for each bond we have the source atom and the target one

These are the functions that we used to generate train and validation data to feed the model. The crucial part consisted on dividing the dataset in batches of the same size and then pad both the remaining rows in the last batch and the dimension of dictionary's values with zeros in order to fit the maximum size (the value zero is taken into account since it has not been used in the tokenization of features).

## 3.2 Model structure

Our first version of the model was mainly as described in figure fig. 3.1.
The inputs of our model consists of an array describing the atoms of the enzymes, an array describing the bonds and a matrix that maintains the connectivity between atoms of the molecules.
Then two **embedding layer** are linked respectively to the atom and bond's input layers. These two layer are nothing more than a matrix of weights and

biases with the shape of the input and with an hidden depth of 128 that characterize the number of feature that we extract from each atom and bond of a given molecule. This determines the size of the model since it influences the size of messages.
Subsequently, we instantiate the first nfp layer that performs the `GlobalUpdate` function receiving in input the two previous embedding layers and the input connectivity matrix. This is essentially an attention layer that looks at the atom and bond states and reduces them to a single vector of length equal to the multiplication between the number of the heads considered for the attention and the output units for each of them. The result is a global state that we will use in the recurrent phase.
As suggested in the paper, we then proceeded by chaining 3 times the nfp layers to perform the message passing phase and create the recurrent unit. The layers to be chained are the `NodeUpdate` and the `EdgeUpdate` followed by a `GlobalUpdate`; all of them take as input the previous atoms state, bond state, global state, and the connectivity matrix. The node update function actually computes the messages to be propagated and use them to update the node hidden state; the second one updates the bonds hidden states and the last one updates the global state that at the end is linked to the output dense layers for the class (qualitative) and pIC50 (quantitative) prediction.

## 3.3 Hyperparameter optimization

Since we have to predict the class of the enzyme so a 0 or 1 value to classify weather or not the enzyme is good (qualitative label), we used a sigmoid as activtion function at the output layer. For what concerns the quantitative parameter, so the effective value of pIC50 for that molecule, we take into account simply the ReLU activation function since value below zero are not concerned and other functions did not contribute to any improvement. For the quantitative prediction task we started using the Mean Absolute Error as loss function that is the typical for regression tasks but after some researches we discovered that the Mean Squared Error was better since in the prediction error it generates higher values causing a stronger update and it is

recommended in medical field. For the loss function of the qualitative prediction we decided to use the binary crossentropy because it optimizes the prediction of the label since it is only 0 or 1.
We have chosen just 1 attention head for the global update function because we did not observe significant improvements in performances increasing the number. The training is performed using a batch size of 64 samples randomly piked shuffling the training set. At the end of each epoch a validation phase if performed. We found the sweet spot for the number of epochs as 250, no significant improvements on metrics was observed training the model more, moreover it would cause overfitting on training set. We also used the Adam optimizer with learning rate of $10^{-3}$ in order to reach efficiently the minimum of the loss functions.

## 3.4 Results

We decided to use two metrics for the qualitative label in order to plot some result and see if the network is learning correctly: accuracy and AUC. The accuracy metric measure the amount of correct prediction over the total while the AUC is a metric that represent the area under the ROC curve and discriminates if the network is predicting label coherently or it just output randomly (a value $\leq 0.5$ means that the network represents a random function or worse).
In this section we report some plot that refer to the best result we obtained over the training sessions. Results for quantitative prediction are measured taking into account the Mean Squared Error due to the consideration made in the previous section. They are both good on training and validation sets, but we have noticed in general a rapid learning at the beginning (first 50 epochs in fig. 3.2) then, while the network continues to improve over training data, it cannot perform better over validation. Considering the fact that the validation set is very small with respect to train and test ones it can be reasonable that it is difficult for the net to enhance the performance.

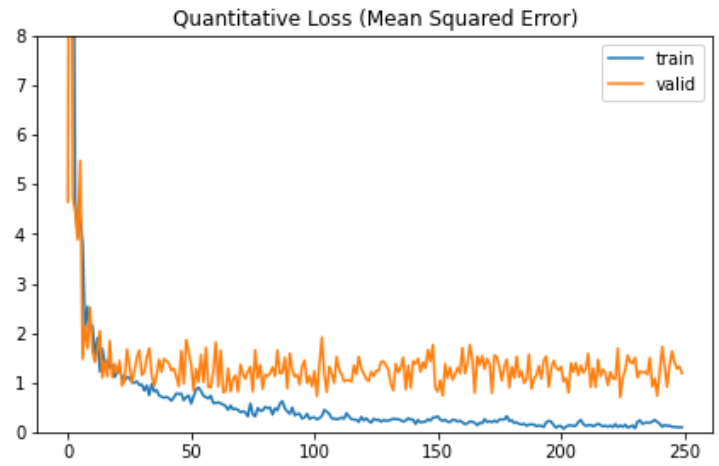


Figure 3.2: Loss function for quantitative prediction

The next metric we take into account is the accuracy score for the qualitative prediction. Also here we can see in most cases that the validation tends to reach in early epochs the highest score and then it is difficult to make better. On the other hand the accuracy over the training set is constantly improving and this makes us expect that the net is learning.

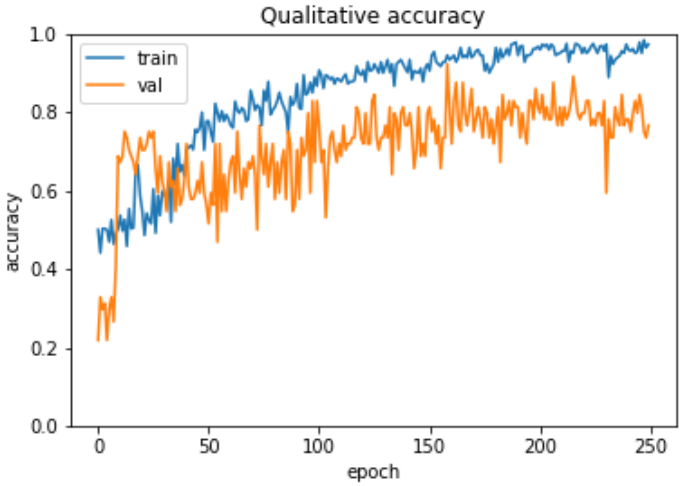


Figure 3.3: Accuracy score for qualitative prediction

Last metric we have taken into account is the AUC (Area Under the Curve). This is, as already said, the real discriminator factor between a good or a meaningless neural network and here we can clearly see the improvement in both cases, thus makes us believe in our previous considerations on problems observed in validation.

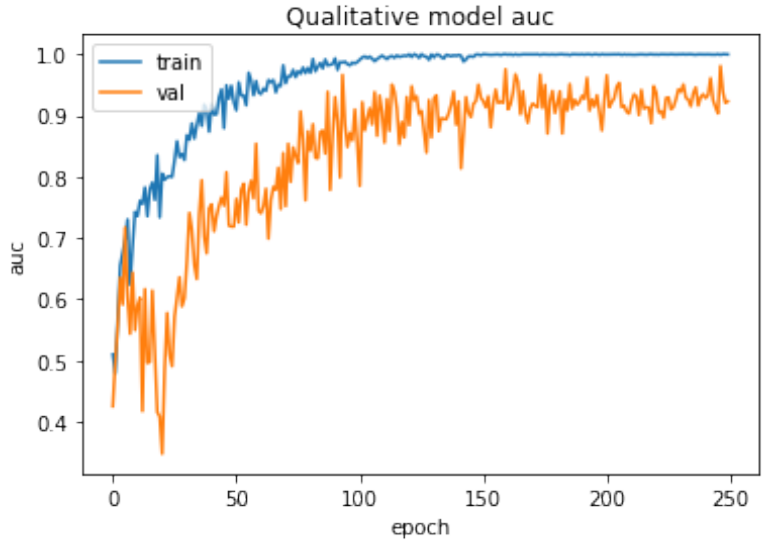


Figure 3.4: AUC value for qualitative prediction

At the end, the scores obtained on test set are the following:

```
Qualitative accuracy on test:   0.6387351751327515
Qualitative AUC on test:        0.7310494184494019
Mean Squared Error on test:     1.5679600238800049
```

Figure 3.5: Final results on test

They do not correspond very well with our expectation considering the plots but anyway they are positive and pretty decent taking into consideration the fact that the size of the test set is at least 6 times the size of the train (around 1200 vs 200).

# Conclusions

We were able to obtain strong performances for our network in the train set but they were not completely satisfying for the validation and test set; to improve our model we probably need a wider dataset to overcome overfitting problems since the network already performs dropout on the dense reduction layers. We think that the performances can be enhanced by changing the method used in the message passing or in the readout phase since the one we used doesn't represent the state of the art; talking about the message function, Glimmer et al. [3] mention a work where the aforementioned function $M_t$ is a neural network itself which takes the concatenation $(h_v, h_w, e_{vw})$ and learns how to generate a better message compared to the matrix multiplication method we used.
Also another big improvement could be done on the $R$ function as proposed by Vynials et al. [5] using the so called Set2Set function. It has been successfully applied to produce a graph embedding representation also in other works for learning molecular representation like the TrimNet [6] and should have more expressive power. The Set2Set aggregates nodes feature by different attention weights and concatenate the aggregated features with history information following this method:

$$q_t = LSTM(q^*_{t-1})$$

$$\alpha_{i,t} = softmax(h_i^T q_t)$$

$$m_\nu^{t+1} = \sum_{i=1}^{N} \alpha_{i,t} h_i^T$$

$$q_t^* = q_t || r_t$$

It performs the above computations for $T$ times recurrently to obtain the final graph representation $q^*$.

We were not able to implement such mechanisms due to high complexity of inserting some new feature in an already designed model. Also it would clearly lead to an increase of the computational cost that probably would make more difficult to train the model in a feasible amount of time.

To investigate the hypothesis of the undersized dataset we swapped the train data and test data since the suggested data for test are 6 times the one originally thought to be used as train set and we obtained some better result (fig. 3.6) showing that it is likely that the main problem here is the size of the dataset.

```
Qualitative accuracy on test:   0.674876868724823
Qualitative AUC on test:        0.7496117353439331
Mean Squared Error on test:     1.8264555931091309
```

Figure 3.6: Final results swapping train and test

# Bibliography

[1] Menting Kelly Willemijn, Claassen Jurgen A. H. R., $\beta$-sretase inhibitor; a promising novel therapeutic drug in Alzheimer's disease, Frontiers in Aging Neuroscience Vol. 6

[2] Jie Zhou, Ganqu Cui, Shengding Hu, et al., Graph neural networks: A review of methods and applications, AI Open

[3] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, George E. Dahl, Neural Message Passing for Quantum Chemistry

[4] Peter C. St. John, Caleb Phillips, Travis W. Kemper, A. Nolan Wilson, Michael F. Crowley, Mark R. Nimlos, Ross E. Larsen, Message-passing neural networks for high-throughput polymer screening

[5] Oriol Vinyals, Samy Bengio, Manjunath Kudlur, Order Matters: Sequence to sequence for set

[6] Pengyong Li, Yuquan Li, Chang-Yu Hsieh, Shengyu Zhang, Xianggen Liu, Huanxiang Liu, Sen Song, Xiaojun Yao, TrimNet: learning molecular representation from triplet messages for biomedicine, Briefings in Bioinformatics