An explanation of the paper - “Convolutional Networks on Graphs for Learning Molecular Fingerprints”

Varun Kumar

BITS Pilani
K.K. Birla Goa Campus

July 15, 2016

What do we want to do?

Extract fixed size feature vector from an arbitrary molecule.

Dataset

The solubility dataset [1] to extract the features and predict solubility of compounds. SMILES representation of molecules were used for training with 'measured solubility in moles per litre' as target labels.

Table : Snapshot of dataset used

Smiles	**Solubility**
OCC3OC(OCC2OC(OC(C#N)c1ccccc1)C(O)C(O)C2O)C(O)C(O)C3O	-0.77
Cc1occc1C(=O)Nc2ccccc2	-3.3
CC(C)=CCCC(C)=CC(=O)	-2.06
c1ccc2c(c1)ccc3c2ccc4c5ccccc5ccc43	-7.87
c1ccsc1	-1.33

Input to the Neural Network

1. Each SMILE is converted to a graph with the nodes representing atoms and edges representing bonds.
2. Handcrafted features are extracted from each node and edge and is called *atom_features* and *bond_features* respectively.
3. A tensor is created of these *atom_features* for all the nodes of graphs/molecules taken in the training set/minibatch.
4. This *input_tensor* of size (#Nodes, length(atom_features)) is used as the input to neural network.

An important data structure

1. array_rep['atom_features'] - The value for this is a matrix of size (#Nodes, length(atom_features)). It stores the node features (of the dataset) sorted according to their degrees.
2. array_rep['bond_features'] - The value for this is a matrix of size (#Edges, length(bond_features)). It stores the edge features (of the dataset) sorted according to their degrees.
3. array_rep['atom_list'] - The value for this is a matrix of size (#SMILES, x); where the value of x is variable and it represents the index of nodes in each SMILE/graph/compound/molecule. Each index points to the index of atom in array_rep['atom_features']

An important data structure

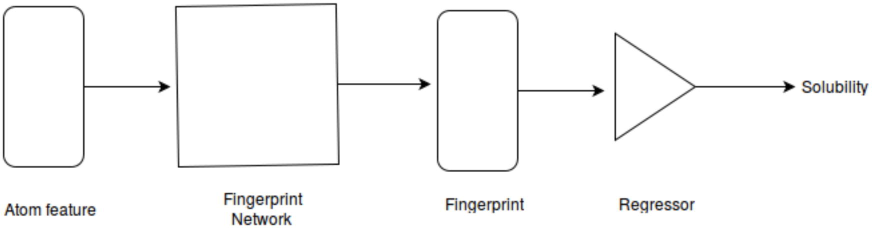
4. array_rep[‘rdkit_ix’] - RDKit, after converting each SMILE to a graph, numbers each node in it (graph) to a unique number (number is unique within the same graph but can be reused among multiple graphs). The value for this dictionary key is an array which contains numerical identifier of the nodes set by RDKit which gives its position in the graph to which each of them belongs.
5. array_rep[(‘atom_neighbors’,**degree**)] - This gives the atom/node neighbors of all the atoms/nodes in the dataset for the particular **degree**. Its size is (#Nodes of degree = **degree** , y); where the value of y is variable and it represents the neighbouring directly bonded nodes.

An important data structure

6. array_rep[('bond_neighbors',**degree**)] - It gives the matrix of size (#Nodes of degree = **degree** , z); where the value of z is variable and it represents the neighbouring bonds (not quite sure about z). See array_rep_from_smiles() in build_convnet.py for implementation details regarding all the above points in this section.

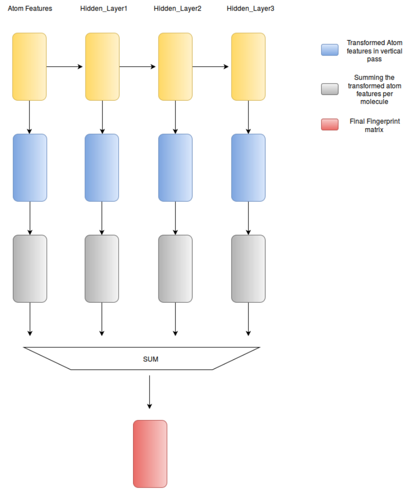
Overall architecture

Figure : A highlevel view of the architecture



Neural fingerprint architecture

Figure : Neural fingerprint architecture



Atom Features
Hidden_Layer1
Hidden_Layer2
Hidden_Layer3
Transformed Atom features in vertical pass
Summing the transformed atom features per molecule
Final Fingerprint matrix
SUM

Methodology - Horizontal Pass

1. At each layer, the atom/node features of previous layer is first transformed by some weight matrix aka *self filter* (which changes the size of the features to size of the next layer), resulting in *self activations*.
2. These *self activations* are summed up with the transformed features of its neighbouring nodes and edges.
3. Now, to find the transformed features of neighbours for each node, first the neighboring nodes and the corresponding edges/bonds of each node is found. Then their feature vectors are found using RDKit. These feature vectors are then concatenated. These concatenated features are then transformed by a weight matrix aka *filter* to give *activations by degree*.
4. These summed activations are then passed through an activation function (relu here) to get the final activations which has to be fed to the next layer.

Methodology - Vertical Pass

1. Input of the current layer is multiplied by output weight matrix aka *layer_output_weights* and is passed through a softmax which results in a fixed size vector of size *FingerPrint_size*. This results in the blue tensors shown in the Figure 2.
2. After multiplyig weight matrix with the set of atom features and getting a huge matrix of size (#Atoms, FP_size) i.e. blue tensors, we then sum up the atom activations molecule-wise. That is for every molecule, sum up the activations of its nodes. This is called *sum_and_stack* operation. This results in a grey tensor of size (#Molecules, FingerPrint_size). We get different grey tensors for each layer (including the Input_Layer).
3. Next, we sum up these tensors/matrices from each layer to get a single Fingerprint matrix of size (#Molecules, FP_size) - red tensor.

Methodology

This fingerprint is treated as input to a linear regression module or another neural network to predict solubility. The root mean squared error in solubility is backpropogated till the start of Fingerprint network to tune the weights of the complete network along with the weights of the linear regressor.

Visualization

1. To get the highest solubility, all the values of fingerprint should be high. For the values of fingerprint should be high, the corresponding atom activations should be high.
2. For each fingerprint **index**, a node/atom before *sum_and_stack* is found out which has the highest activation value for that **index**.
3. Parent molecule is found out is found out for that node.
4. Molecule is ploted and the atom selected in point 2. is highlighted.

Bibliography

[1] John S Delaney. Esol: estimating aqueous solubility directly from molecular structure. *Journal of chemical information and computer sciences*, 44(3):1000–1005, 2004.