An explanation of the paper - “Neural fingerprint generation of molecules”

Varun Kumar
BITS-Pilani, K.K. Birla Goa Campus

July 5, 2016

Contents

0.1 Introduction

This report explains the working of the code from the paper "Convolutional Networks on Graphs for Learning Molecular Fingerprints". Though the title of the paper says "Convolution Networks", the network that thay have constructed is quite different from the traditional Covolution Networks which we have come across while working with images or audio signals. Author calls it convolution network because same operation is performed over all the atoms in a molecule, this is similar to using the same filter on multiple places of the image. As the next layer of the convolutional network gets a larger receptive field of the image as compared to the previous layer, here also, with every new layer(or radius) the receptive field of the graph on the atoms is increased. I have considered the molecular package RDKit used in the paper as a black box. It was used to construct graph from SMILES representation of molecules and extract the handcrafted features of atoms and bonds. They formulated the problem of extracting fixed sized features as a regression problem for predicting solubility which generates these fixed sized features in between.

0.2 Dataset

They have used the solubility dataset [1] to predict solubility. SMILES representation of molecules were used for training with 'measured solubility in moles per litre' as target labels. A snapshot of the dataset used is shown in the Table 1. Each SMILE converted to a graph with the nodes representing atoms and edges representing bonds. Handcrafted features are extracted from each node and edge and is called *atom feature* and *bond feature* respectively. A tensor is created of these atom features for all the nodes of graph of the molecules taken in the training set/minibatch. This *input tensor* of size (#Nodes, length(atom features)) is used as the input to neural network. Hence each atom becomes a unique node i.e. the same carbon atom present in different molecules will be referred to as different nodes.

Table 1: 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

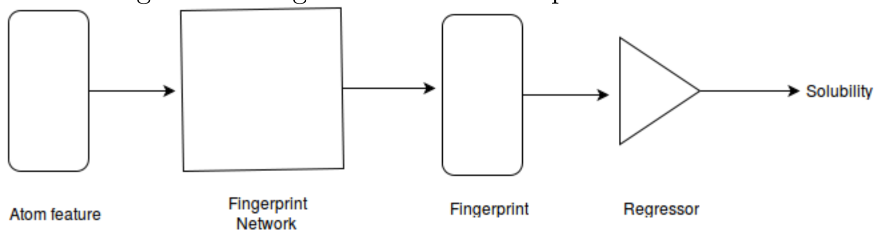
0.3 Architecture

They created the architecture to extract a fixed size fingerprint of a molecule and then used it to predict its solubility using another neural network. In place of a full neural network a linear regressor can also be used (See figure 1) for the task of prediction. They trained the network end to end thereby generating the features in between. I like to call this architecture a 2 way neural network with a horizontal pass and a vertical pass.

A 4 layered neural net is used for feature extraction. Layers are - [Input_Layer, Hidden_Layer1, Hidden_Layer2, Hidden_Layer3]. The layer and radius is used interchangably in this report. *Input tensor* is forward propogated via these layers described above. - I call this a horizontal pass. The reason behind forward propogating the whole tensor at once is that after every layer you need to update the features of all the atom nodes in the tensor together. Because at each update, the atom features are summed up with the neighbouring atom features i.e. after every layer/radius the atom node contains a representation of its neighbouring atom nodes as well. Therefore it is manditory that all the neighbouring nodes have the features of the same layer.

Now you may wonder where is the output layer. There is no single output layer like a traditional neural network has. Instead, output is calculated at each layer (including the Input_Layer). - I call this the vertical pass.

Figure 1: A high level view of complete architecture



0.4 Methodology

Horizontal Pass:

In horizontal pass, outputs of the previous layer is updated and tranformed as inputs of the next layer. So there are 3 Such transformations as per the architecture shown in the previous section. Let us look at what happens at each transformation.

- At each layer, the atom 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.*

- These *self activations* are summed up with the transformed features of its neighbouring atoms and bonds.

- Now, to find the transformed features of neighbours for each atom node, first the neighboring atom nodes and corresponding bonds of each atom node is found. Then their feature vectors are found using RDKit. These feature vectors are then concatenated.

- These concatenated features are again transformed by another weight matrix aka *filter* to give *activations by degree.* Degree of an atom is defined to be its number of directly-bonded neighbors. As any atom can have maximum of 5 degrees (for organic molecules), we need 5 different *fiters.*

- Another point to note here is that the list of atoms is sorted according to the degree of atoms. The top atom in the list is the atom of degree 0 or 1 and the last atom is of maximum degree of all the atoms in the dataset. For each degree corresponding set of atoms are selected and their neighbours are found out. Then the neighbours belonging to a particular degree atom is multiplied by a particular *filter* to get *activations by degree* as mentioned in the previous point. This is done for all the degrees. The results are then appended one after another to get the transformed neighbour activations for each atom in the list.

- As mentioned eatlier in the second point, these transformed neighbour activations are then added to the *self activations* to get the summed activations.

- These summed activations are then passed through an activation function (here relu) to get the final activations to be fed to the next layer.

Vertical Pass:

- For vertical pass, 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. This results in the blue tensors shown in the Figure 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 subsets of atom activations that belong to each seperate molecule. This is called *sum_and_stack* operation. This results in a grey tensor of size (#molecules, FP_size). We get this type of different grey tensors for each layer (including the Input_Layer).

- Next, we sum up these tensors/matrices from each layer to get a single Fingerprint matrix of size (#molecules, FP_size) - red tensor. This is our final fingerprint output which we are going to improve later by updating the weights of the“convolution” aka fingerprint network by gradient descent.

So, to summarize, we can say that in total we deal with 3 types of weight matrices.

- Self filter (One for each transformation among layers)
- Filter (One for each degree)
- Output Weight matrix (One for each layer including the Input_Layer)

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. The detailed view of fingerprint network that extracts fixed size fingerprints for each molecule can be seen in Figure 2.

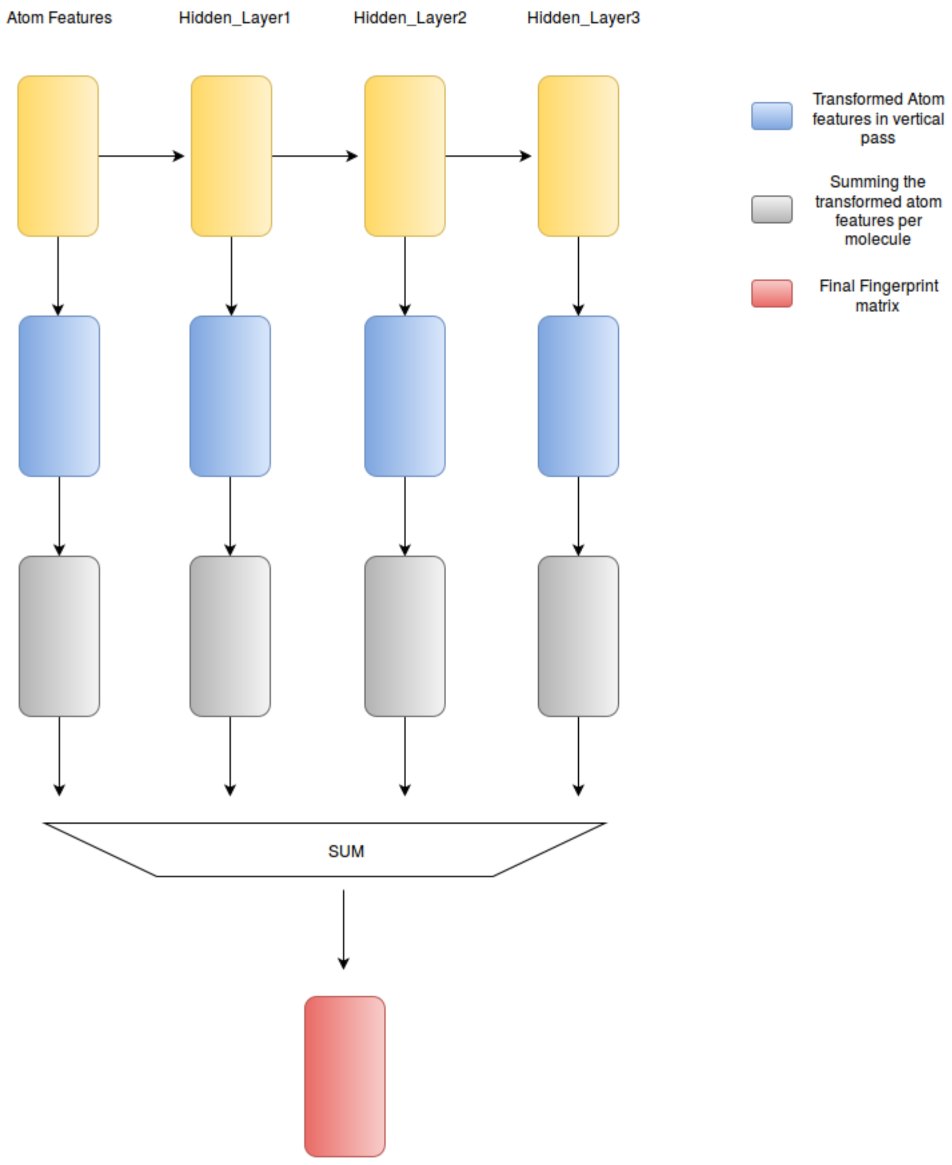
0.5 Visualization

This section describes about the intrepretability of the features learned. To visualize which atoms contriubute most towards solubility, they tried to find out the atoms that have the highest activations among all the layers of the fingerprint network. Before *sum_and_satck* operation they appended the blue tensors shown in the figure 2 into one long list. They then calculated, for each index of fingerprint, which atom for the the same index as of fingerprint, has the highest activation. Then highlighted that atom. The neighbours are also highlighted based on the radius of the atom under consideration. If radius is 0, the atom itself is highlighted. If radius is 1, the atom and its neighbours are highlighted. If radius is 2, then the atom, its neighbours and its neighbours’ neighbours are highlighted and so on. Radius refers to the layer of neural net the atom with the highest activation belonged to.

I think what they mean is that to get higher solubility, all the values of fingerprint should be high. For the values of fingerprint should be high, the corresponding atom activations should be high.

They then plotted the atoms and highlighted them that contributed highest to the solubility.

Figure 2: A detailed level view of Fingerprint Network


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

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.