BITS Pilani
K K Birla Goa Campus

innovate achieve lead

Thesis

Sem-1 Year 2019-2020

“Developing Prediction tools for predicting Physico-chemical and Quantum Mechanical properties of small molecules”

Department of physics

By:2013B5A80458G

KESANI MEHER MANOJ

Acknowledgement

I would like to thank the instructor in-charge of my thesis Dr. Ravipasad Aduri for giving me an opportunity to work on this topic and for providing me with all the help. I would also like to thank the Department of Biological Sciences and my university Bits-Pilani, Goa campus.

Abstract

For a given a dataset of molecules and their properties, we explore through different properties (that are intrinsic to the structure of our molecule) when represented in a graphical model.Using these properties as an input we develop machine learning algorithms that reliably predict the properties in the dataset.

Introduction

The basic idea is to represent a molecule as a graph structure,using this as input to message passing neural network to predict the outcome. Information about the molecule is represented in structural format ie., in Space domain.We convert this into time domain so that the substructure we obtain after few iterations keeps the maximum amount of information about the molecule, essentially reducing the dimensionality. Using these forms for representing a molecule we predict the properties in our dataset.

The neural network should be invariant to graph isomorphism.This give us the same output irrespective of the order the vertices in the input.Here the Smiles format of a molecule(adjacency matrix) is used as an input to the network,which mostly have “0” as their entries.These kind of matrices are called Sparse matrices.The dataset we use is QM9 which has properties such as geometries minimal in energy, corresponding harmonic frequencies, dipole moments, polarizabilities...which were developed using Density Functional Theory.

1 Pure Literal Elimination and Random K Satisfiability

To work with sparse matrix we have to reduce the dimensionality of the matrix,the following is the analysis of the process using a message passing network.

Pure Literal: A variable which always appears as a directed value or always appears as negated value is called a pure literal. If we have variable that always appears negated we set it to False.

Consider a uniformly random formula with ‘n’ variables(V) and ‘nα’ clauses (E) represented as a labeled bipartite graph with n variables on one side and nα on the other. First we reduce this with random choices which forms a markov chain in reduced space,then study the ODE obtained.

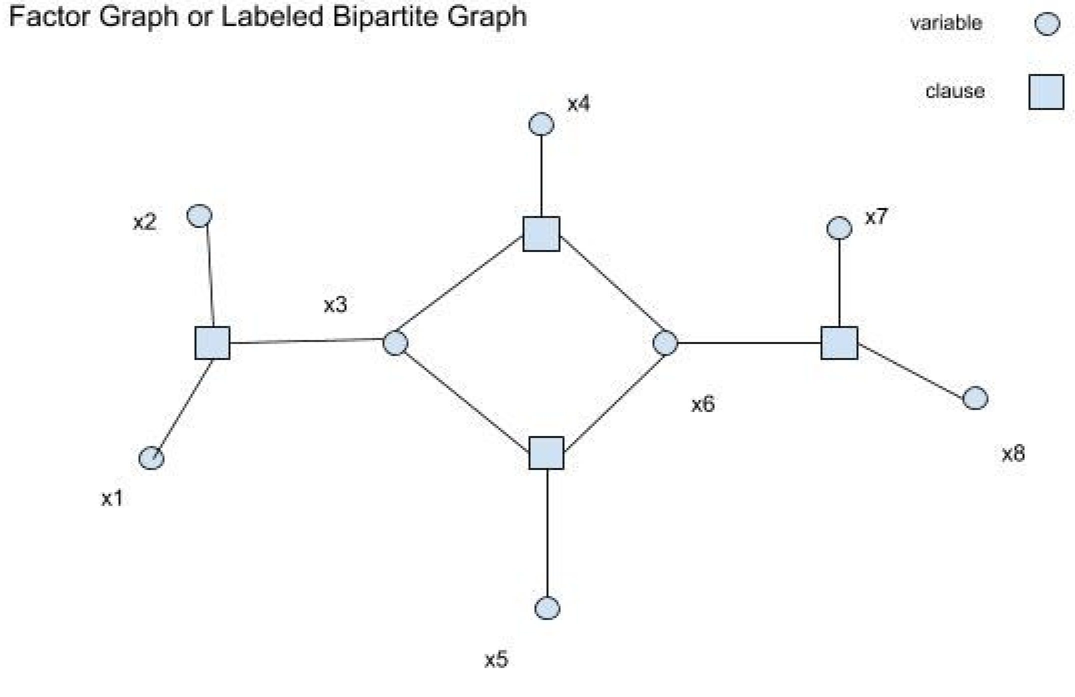
Pure literal elimination

1. Repeat
2. Find pure literals (x)
3. Fix x ;

Let us select a random satisfiability formula and reduce it with random choices using message passing network.

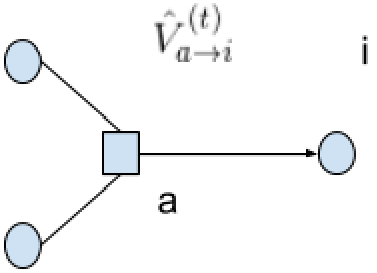
2 Message Passing

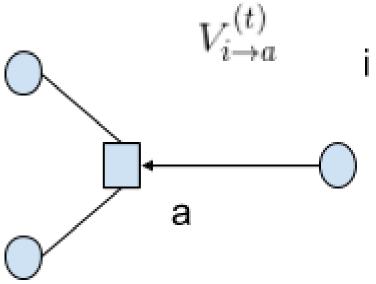
2.1 Factor Graph:A random formula is represented as factor graph (G) in the following figure.Message can be passed between variables and clauses.



eg:$(x1 \vee x2 \vee x3') \wedge (x3 \vee x4' \vee x6') \wedge (x3 \vee x5' \vee x6) \wedge (x6 \vee x7 \vee x8')$

2.2 Message association: Let v_{ia} be the message sent from variable ‘i’ to the clause ‘a’ and $\widehat{v}_{ai}$ from clause to variable which can be interpreted as free or constrained.





(v_{ia}, $\widehat{v}_{ai}$) ∈ $\mathcal{M}(\{0,1\})$ (simplex of probability measure on{0,1})

Consider a message from clause

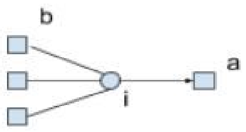
- Free: To satisfy me you are free to take any value.
- Cons: To satisfy me you are constrained to take a certain value.

Similarly a message from variable

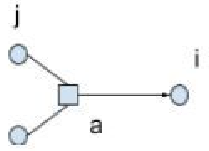
- Free: I am free to satisfy you.
- Cons:I am not free to satisfy you.

A variable is always constrained unless it is free with respect all its other cluses. This can be considered as the initial condition for our system.

2.3 Update Rule:Now let us see how the update of the network is done in each time step.



$$V_{i\to a}^{(t+1)}(x_i) = \prod_{b\in\delta i\setminus a} \hat{V}_{b\to i}^{(t)}(x_i)|$$



$$\hat{V}_{a\to i}^{(t)}(x_i) = 1 - \prod_{j\in\delta a\setminus i} V_{j\to a}^{(t)}(x_j = s_{ja}) if (x_i = s_{ia})$$

$$=1 \qquad \text{otherwise}$$

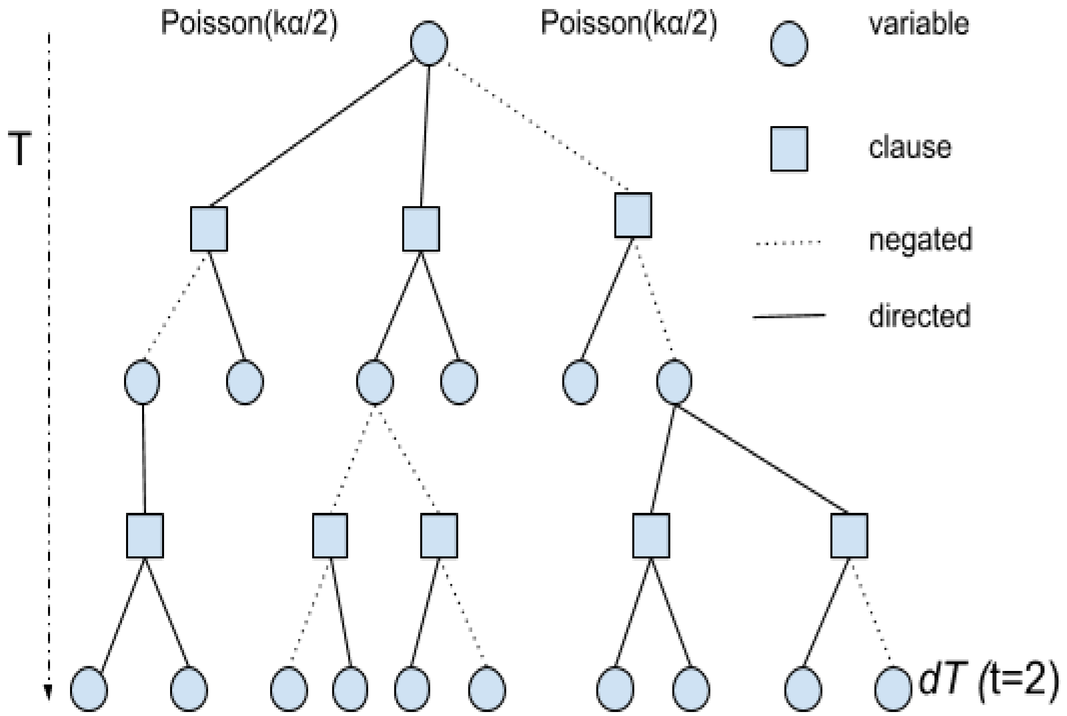
S_{ai} is the label of the edge between ‘a’ and ‘i’. The message to clause ‘a’ from variable ‘i’ will be a product of all the messages ‘i’ got except the one from ‘a’. $\widehat{v_{ai}}$ will change only if the variable value is equal to the corresponding label otherwise it remains ‘1’. This change will be the product of all the messages clause received except for ‘i’.

After ‘t’ iterations we will be collecting the information from a ball of radius ‘t’ (B(l,t),l ∈ $[n]$). Now let’s see how the density of the information evolves over time.Consider a random edge of the graph(G) where the variable is connected to ‘d’ clauses and the clause is connected to ‘k’ variables.Let $\Phi(t)$ be the probability of v_{ia} being constrained and $\widehat{\Phi(t)}$ be the probability of $\widehat{v_{ai}}$ being constrained.

The sequences of (factor) graphs G =(V=[n],E) converges locally to a random rooted tree 'T' if, for any 't' B(t) converges in distribution to T(t) the first 't' generations of the tree.

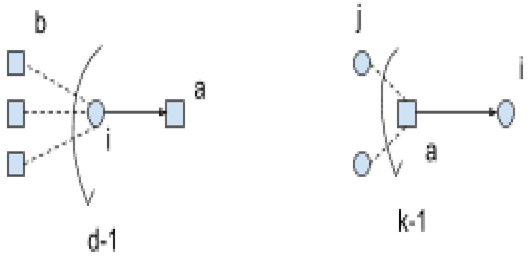
In the graph G pick a random vertex as the root of the random tree 'T' with poisson distribution (p($k\alpha/2$)) as the vertex degree distribution.

For this infinite tree T(∞) let μ^t(x) be the uniform measure over the solution.



2.4 Gibbs Measure & Density Evolution: A measure is uniform or gibbs if the conditional distribution upto depth ‘t’ conditional to that of value at depth ‘dT’,is uniform for any depth. That is μ^t(x(T^t)|x(d T^t)) is uniform for any ‘t’.Furthermore we gain an advantage that is as t approaches the uniqueness threshold also called pure literal’ $\alpha_{pl}(k)$’, the influence of this vertex on the root decays exponentially.

$$\mathrm{SUP}_{x,x'}\left|[\ \mu^t(x(T^1)|x(d\ T^t))]-[\ \mu^t(x(T^1)|x'(d\ T^t))]\right| \leq Ae^{-bt}$$



$$\Rightarrow \Phi(t+1) \quad = \quad 1\text{- Expectation}\{[1-(\widehat{\Phi}(t)\ /2)]^{d}\}$$

$$\Rightarrow \widehat{\Phi}(t) \quad = \quad [\ \Phi(t)\ ]^{k-1}$$

The degree distribution of the node is asymptotically poisson. So we have

$$\Rightarrow \Phi(t+1) \quad = \quad 1\text{- Exp}(-k\,\alpha\,[\widehat{\Phi}(t)]^{k-1}/2\ )$$ with initial condition Φ_1 = 1.

$$\Rightarrow Y \ = 1\text{- Exp}(-k\,\alpha\, x^{k-1}/2\ )$$

we see that our tree T converges similarly to that of Poisson-Galton-Trees,meaning for $\alpha < \alpha_{pl}(k)$ our solution will converge freeing all the independent variables.

Conjecture : For a gibbs sampler the mixing time polynomial $\tau = O(n^c)$ for $\alpha < \alpha_u$

Finally let’s consider the uniqueness of the solution.If we take a random K-sat the Gibbs measure is unique if and only if:

$\alpha < \alpha_u$ with α_u =2log(k)\k[1+ o_k (1)]

$\alpha_{pl}(k)$ = α_u + O(k^{-2})

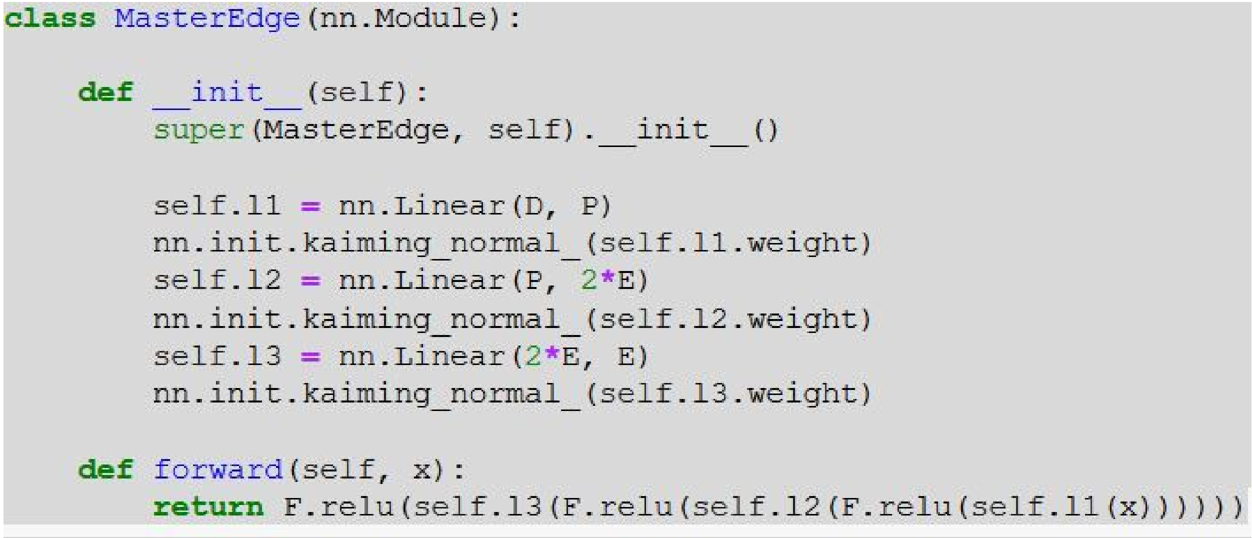
So, we took our input and represented it as a graph, using message passing as random choices to reduce the state space to a finite depth rooted tree which is unique.But we are using this as a machine learning model so we should consider the factors which affect the learning rate and predictability.

- A healthy amount of data is required as we are representing high dimensional input as compressed data to the network as to avoid missing out on the important dimensions .

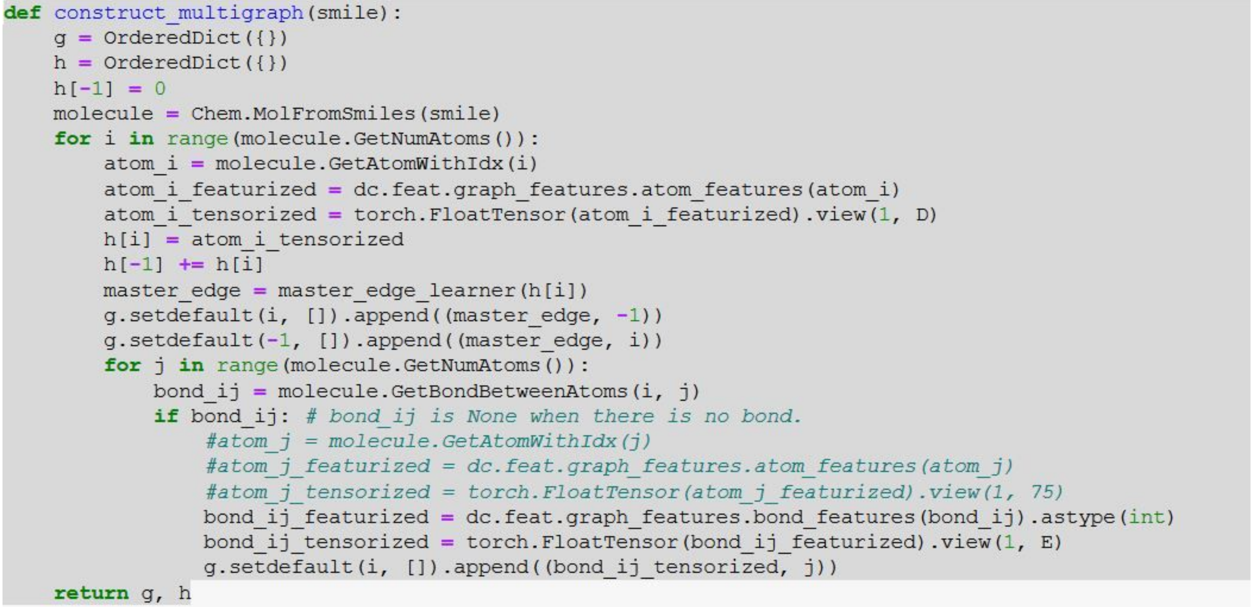
- The loopiness of the initial structure may require large number of iterations to pass the message.

3 Approximate *Python* Code:

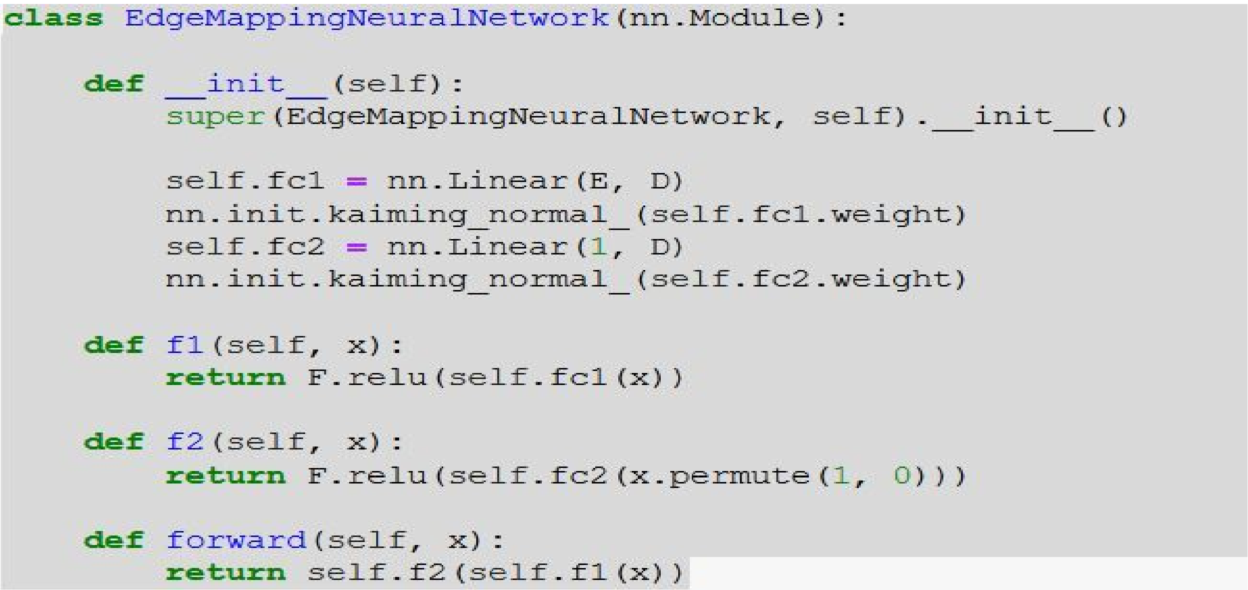
```
import os
import numpy as np
import pandas as pd
from collections import OrderedDict
import deepchem as dc
from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem
from deepchem.feat.graph_features import *
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
N  # number of molecules in the dataset
D  # hidden dimension of each atom
E  # dimension of each edge
T  # number of time steps the message phase will run for
P  # dimensions of the output from the readout phase, the penultimate output
V  # dimensions of the molecular targets or tasks
qm9 = pd.read_csv('qm9.csv')
structures = ['smiles']
tasks = ['mu', 'alpha', 'homo', 'lumo', 'gap', 'r2', 'zpve', 'u0', 'u298', 'h298', 'g298', 'cv']
from sklearn.preprocessing import MinMaxScaler, RobustScaler, StandardScaler
scaler = StandardScaler()
y = pd.DataFrame(scaler.fit_transform(y), index=y.index, columns=y.columns)
def batch_mse_loss(pred, true):
        return F.mse_loss(pred, true) / BATCH_SIZE
def valid_mse_loss(pred, true):
        return (F.mse_loss(pred, true)).detach() / VALID_SIZE
```



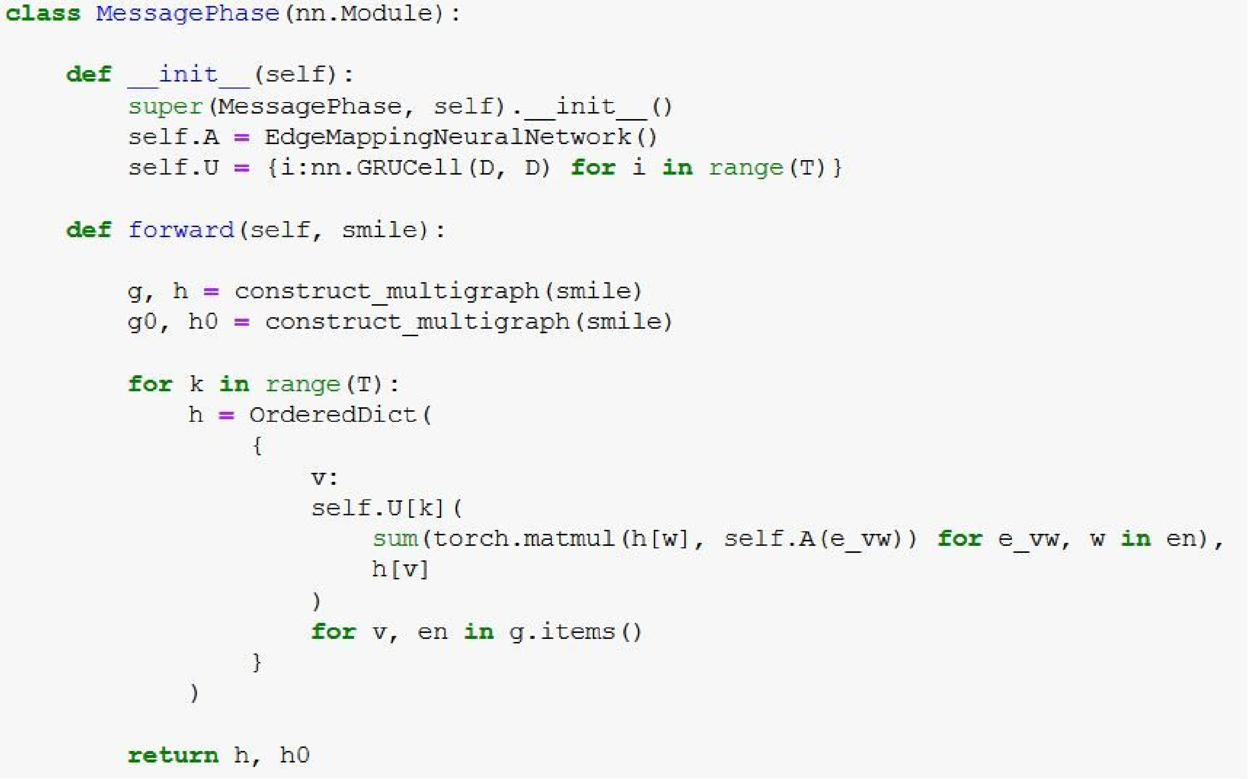
```
class MasterEdge(nn.Module):

    def __init__(self):
        super(MasterEdge, self).__init__()

        self.l1 = nn.Linear(D, P)
        nn.init.kaiming_normal_(self.l1.weight)
        self.l2 = nn.Linear(P, 2*E)
        nn.init.kaiming_normal_(self.l2.weight)
        self.l3 = nn.Linear(2*E, E)
        nn.init.kaiming_normal_(self.l3.weight)

    def forward(self, x):
        return F.relu(self.l3(F.relu(self.l2(F.relu(self.l1(x))))))
```



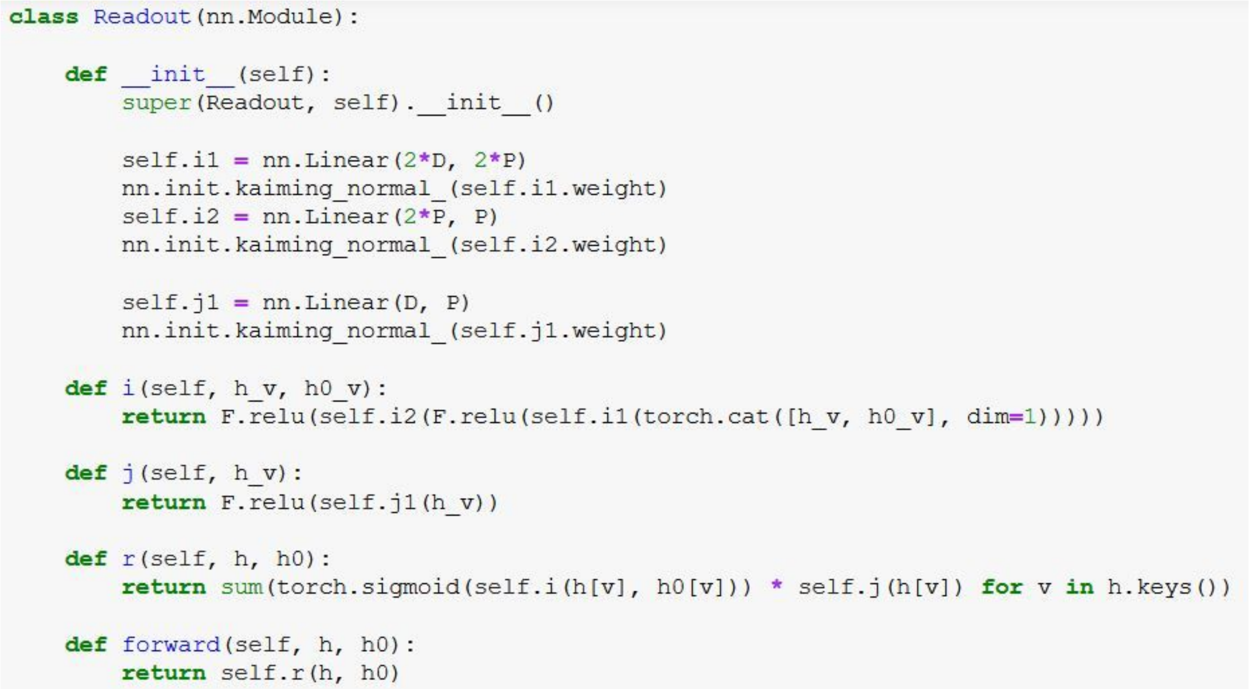
```
def construct_multigraph(smile):
    g = OrderedDict({})
    h = OrderedDict({})
    h[-1] = 0
    molecule = Chem.MolFromSmiles(smile)
    for i in range(molecule.GetNumAtoms()):
        atom_i = molecule.GetAtomWithIdx(i)
        atom_i_featurized = dc.feat.graph_features.atom_features(atom_i)
        atom_i_tensorized = torch.FloatTensor(atom_i_featurized).view(1, D)
        h[i] = atom_i_tensorized
        h[-1] += h[i]
        master_edge = master_edge_learner(h[i])
        g.setdefault(i, []).append((master_edge, -1))
        g.setdefault(-1, []).append((master_edge, i))
        for j in range(molecule.GetNumAtoms()):
            bond_ij = molecule.GetBondBetweenAtoms(i, j)
            if bond_ij: # bond_ij is None when there is no bond.
                #atom_j = molecule.GetAtomWithIdx(j)
                #atom_j_featurized = dc.feat.graph_features.atom_features(atom_j)
                #atom_j_tensorized = torch.FloatTensor(atom_j_featurized).view(1, 75)
                bond_ij_featurized = dc.feat.graph_features.bond_features(bond_ij).astype(int)
                bond_ij_tensorized = torch.FloatTensor(bond_ij_featurized).view(1, E)
                g.setdefault(i, []).append((bond_ij_tensorized, j))
    return g, h
```



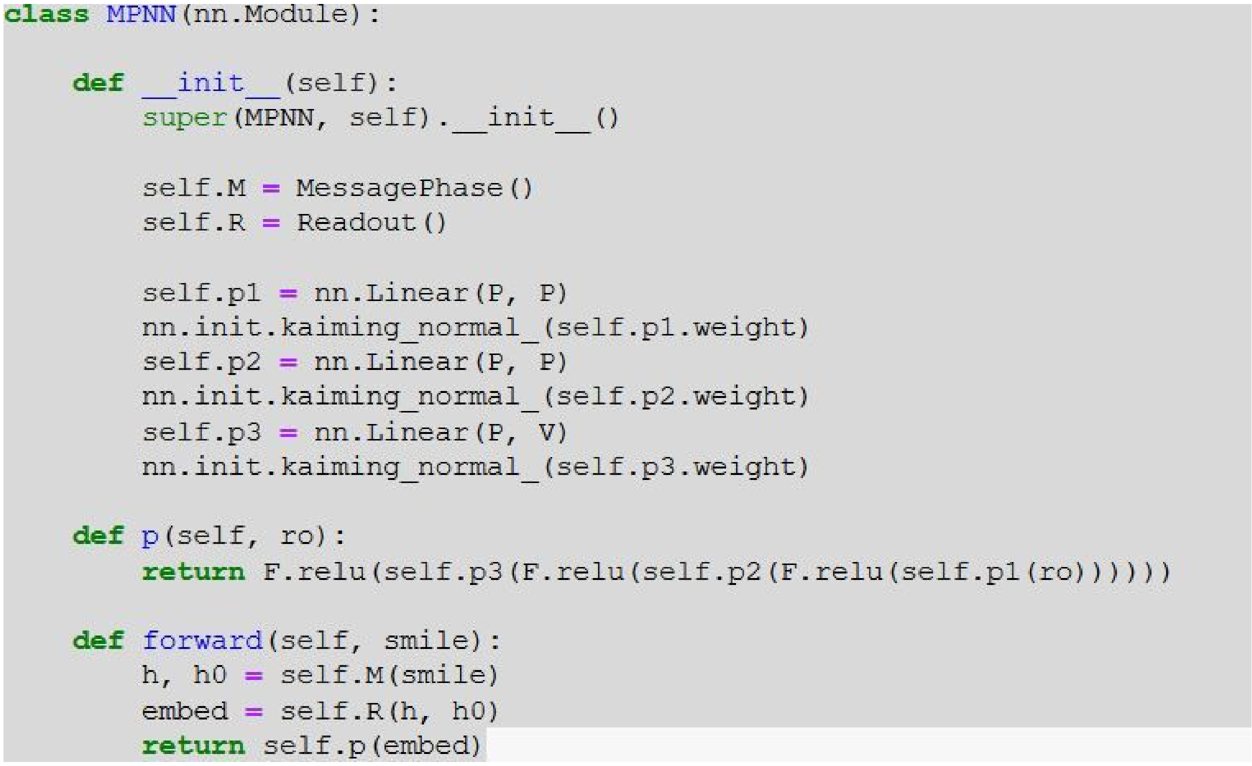
```
class EdgeMappingNeuralNetwork(nn.Module):

    def __init__(self):
        super(EdgeMappingNeuralNetwork, self).__init__()

        self.fc1 = nn.Linear(E, D)
        nn.init.kaiming_normal_(self.fc1.weight)
        self.fc2 = nn.Linear(1, D)
        nn.init.kaiming_normal_(self.fc2.weight)

    def f1(self, x):
        return F.relu(self.fc1(x))

    def f2(self, x):
        return F.relu(self.fc2(x.permute(1, 0)))

    def forward(self, x):
        return self.f2(self.f1(x))
```



```
class MessagePhase(nn.Module):

    def __init__(self):
        super(MessagePhase, self).__init__()
        self.A = EdgeMappingNeuralNetwork()
        self.U = {i:nn.GRUCell(D, D) for i in range(T)}

    def forward(self, smile):

        g, h = construct_multigraph(smile)
        g0, h0 = construct_multigraph(smile)

        for k in range(T):
            h = OrderedDict(
                {
                    v:
                    self.U[k](
                        sum(torch.matmul(h[w], self.A(e_vw)) for e_vw, w in en),
                        h[v]
                    )
                    for v, en in g.items()
                }
            )

        return h, h0
```



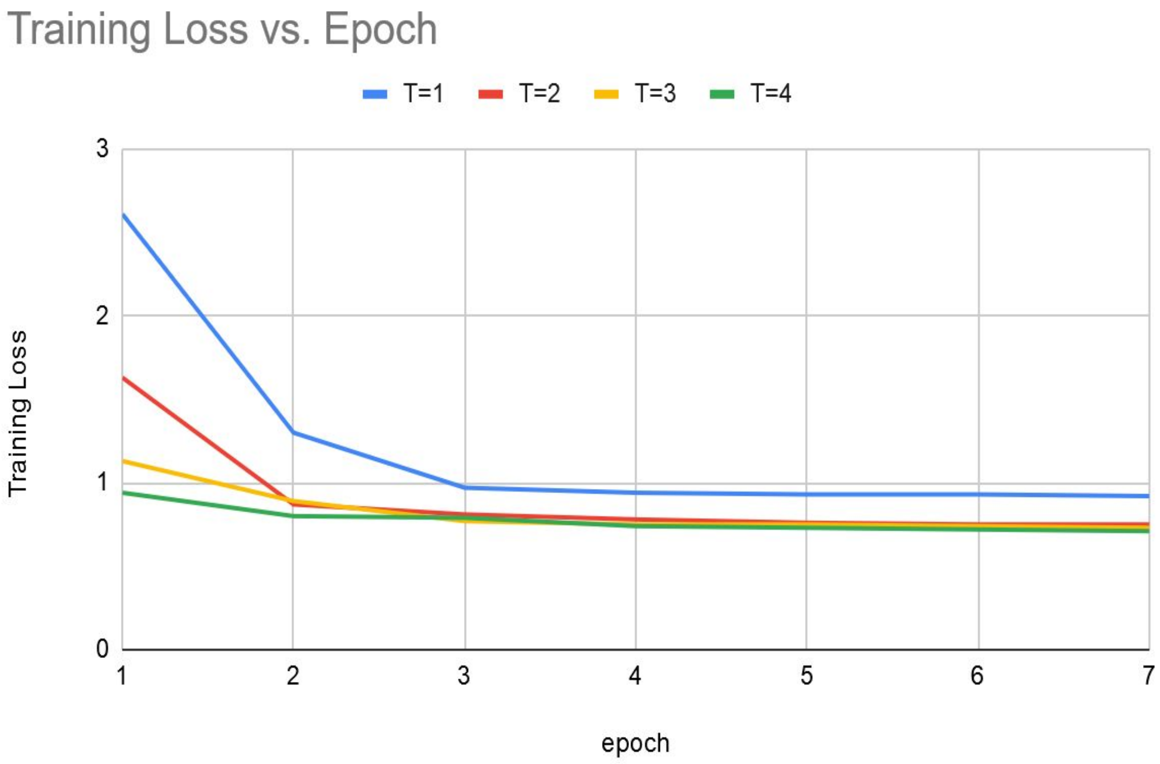
```
class Readout(nn.Module):

    def __init__(self):
        super(Readout, self).__init__()

        self.i1 = nn.Linear(2*D, 2*P)
        nn.init.kaiming_normal_(self.i1.weight)
        self.i2 = nn.Linear(2*P, P)
        nn.init.kaiming_normal_(self.i2.weight)

        self.j1 = nn.Linear(D, P)
        nn.init.kaiming_normal_(self.j1.weight)

    def i(self, h_v, h0_v):
        return F.relu(self.i2(F.relu(self.i1(torch.cat([h_v, h0_v], dim=1)))))

    def j(self, h_v):
        return F.relu(self.j1(h_v))

    def r(self, h, h0):
        return sum(torch.sigmoid(self.i(h[v], h0[v])) * self.j(h[v]) for v in h.keys())

    def forward(self, h, h0):
        return self.r(h, h0)
```

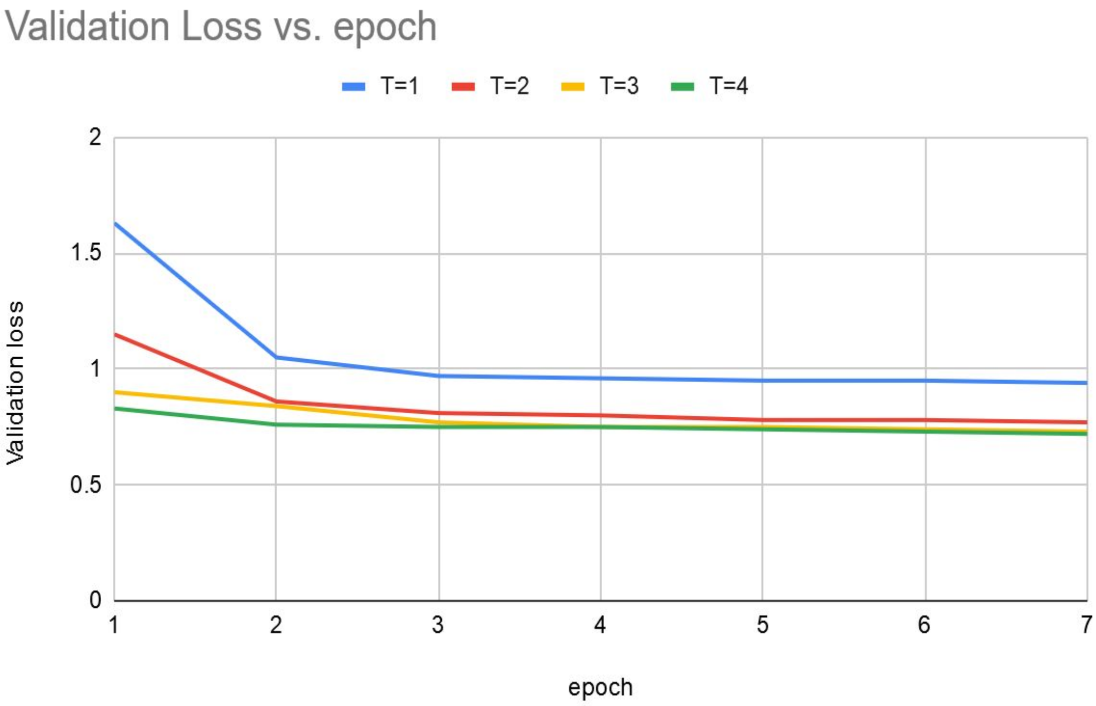


```
class MPNN(nn.Module):

    def __init__(self):
        super(MPNN, self).__init__()

        self.M = MessagePhase()
        self.R = Readout()

        self.p1 = nn.Linear(P, P)
        nn.init.kaiming_normal_(self.p1.weight)
        self.p2 = nn.Linear(P, P)
        nn.init.kaiming_normal_(self.p2.weight)
        self.p3 = nn.Linear(P, V)
        nn.init.kaiming_normal_(self.p3.weight)

    def p(self, ro):
        return F.relu(self.p3(F.relu(self.p2(F.relu(self.p1(ro))))))

    def forward(self, smile):
        h, h0 = self.M(smile)
        embed = self.R(h, h0)
        return self.p(embed)
```

```
model = MPNN()
optimizer = optim.Adam(model.parameters(), lr=LR)
```

4 Conclusion: The number of message passing step is our hyperparameter(T).There is no significant development for T>4. Following are graphs showing the training loss and validation loss.





5 References

1. On the Satisfiability and Maximum Satisfiability of Random 3-CNF Formulas by **Andrei Z. Broder and Alan M. Frieze**
2. Random walks and harmonic functions on infinite planar graphs using square tilings by **ITAI BENJAMINI AND ODED SCHRAMM**
3. The Objective Method: Probabilistic Combinatorial Optimization and Local Weak Convergence by **David Aldous J. Michael Steele**
4. Tree-valued Markov chains derived from Galton-Watson process by **David ALDOUS and Jim PITMAN**
5. Counting good truth assignments of random k-SAT formula by **Andrea Montanari and Devavrat Shah**
6. Gibbs measures and phase transitions in sparse random graphs by **Amir Dembo and Andrea Montanari**
7. Monte Carlo Methods for Randomized Likelihood Decoding by **Alankrita Bhatt, Jiun-Ting Huang**

8. Low-Density Parity-Check Code by **Robert G Gallager**
9. Convolutional Networks on Graphs for Learning Molecular Fingerprints by **David Duvenaud, Dougal Maclaurin**
10. Neural Message Passing for Quantum Chemistry by **Justin Gilmer, Samuel S. Schoenholz**
11. Self-Normalizing Neural Networks by **Günter KlambauerThomas Unterthiner**
12. Empirical Evaluation ofGated Recurrent Neural Networks on Sequence Modeling by **Junyoung Chung and Caglar Gulcehr**
13. Quantum chemistry structures and properties of 134 kilo molecules by **Raghunathan Ramakrishnan,Pavlo O. Dral,Matthias Rupp**