#2 Solution: SchNet and SchNet with Edge Updates

Xinyu Li, The University of Newcastle, Australia

Email: Xinyu.Li@uon.edu.au

November 17, 2019

I would like to thank Tencent Quantum Lab for organizing such an awesome competition. Here's a brief description of the method I used for this #2 entry.

1 Introduction

First, a quick introduction of my background. I am a PhD student at the University of Newcastle, Australia. My research topic is using machine learning to predicting energy materials (catalysts) which can transfer biomass into hydrogen. Thus I have background at both Quantum Chemistry and Machine Learning. However, most of my previous work focused on feature engineering which using the state-of-art representations (e.g. SOAP[1], SLATM[2], FCHL[3]) to predict those quantum properties. I did not have many experience in using Deep Learning so I did not expect that I would end at rank # 2 solution, which was a big surprise for me.

Before this competition, there was a Kaggle Competition named "Predict Molecular Properties". Although that competition was predicting scalar coupling constant, but the common ideas and methods were very similar. I read lots of discussions there and decided to use SchNet[4].

2 Packages

First I tried the SchNet[4, 5] implemented in SchnetPack[6]. I got very low MAE in predicting Property 0, 3, 5, 7, 8, 11 (lower than 1e-3, which makes them were unimportant for the overall MAE). Then I would like to move to SchNet with Edge Updates[7]. However, SchnetPack was too difficult for me to modify it, but luckily I found Tencent Alchemy Github Repo provide a version based on DGL, so my modification was based on the SchNet model on Tencent Alchemy Github Repo.

3 Feature Selections

Only atomic numbers and Cartesian coordinates are used. Lots of discussions and reference[8, 9] claimed that adding bond features such as bond type improved performance. However, in my test I did not see any improvement.

4 Data Preparation

SchnetPack only accept Atomic Simulation Environment[10] db as input file, thus all input sdf files were transferred into a ase db file by the following script.

```
import pandas as pd
import numpy as np
import os
from ase import io

from multiprocessing import Pool
import pathlib
from schnetpack.data import AtomsData
from tqdm import tqdm
import shutil

dirname = os.getcwd()
datadir = os.path.join(dirname, 'data-bin', 'raw')
train_dir = os.path.join(datadir, 'dev', 'sdf')
valid_dir = os.path.join(datadir, 'valid', 'sdf')
test_dir = os.path.join(datadir, 'test', 'sdf')

properties = ['property_%d' % x for x in range(12)]

from rdkit import Chem
from rdkit.Chem import AllChem
from ase import Atom, Atoms
from ase.io import write

def sdf_2_ase_atoms(file):
    '''
    the function to transfer a sdf file to a ase atoms
    Inputs:
        file    The file path of a sdf file
    '''
    with open(file, 'r') as f:
        sdf_string = f.read()
    mol = Chem.MolFromMolBlock(sdf_string, removeHs=False)
    xyz = np.empty((mol.GetNumAtoms(), 5))
    xyz = xyz.astype(str)
    conf = mol.GetConformer()
    ase_atoms = []
    for i in range(conf.GetNumAtoms()):
        position = conf.GetAtomPosition(i)
        atom = mol.GetAtoms()[i]
        xyz[i, 0] = atom.GetSymbol()
        if position.x >= 0:
            xyz[i, 1] = "%0.5f"% position.x
        else:
```

```
            xyz[i, 1] = "%0.4f"% position.x
        if position.y >= 0:
            xyz[i, 2] = "%0.5f"% position.y
        else:
            xyz[i, 2] = "%0.4f"% position.y
        if position.z >= 0:
            xyz[i, 3] = "%0.5f"% position.z
        else:
            xyz[i, 3] = "%0.4f"% position.z
        ase_atoms.append(Atom(symbol=xyz[i, 0], position=(xyz[i, [1, 2, 3]])))
    ase_molecule = Atoms(ase_atoms)


    return ase_molecule

# Generate training atoms db

df = pd.read_csv(os.path.join(datadir, 'dev','dev_target.csv'),
                 index_col=0,
                 usecols=['gdb_idx',] +['property_%d' % x for x in range(12)])
train_structures = []
sdf_dir = pathlib.Path(train_dir)
property_list = []
i = 0
ind = []
for sdf_file in tqdm(sdf_dir.glob("**/*.sdf")):
    i += 1
    name = str(sdf_file).split('/')[-1].split('.')[0]
    ind.append(int(name))
    atoms = sdf_2_ase_atoms(sdf_file)
    train_structures.append(atoms)
    properties = {n : np.array([df.loc[int(name), n]]).astype(np.float32) for n in ['property_%d' %
    property_list.append(properties)

print("get {} items".format(i))

new_dataset = AtomsData('train_dataset.db', available_properties= properties)#['property_%d' % x f
new_dataset.add_systems(train_structures, property_list)
```

For the input of the SchNet implemented in Tencent Alchemy Git Repo, all the sdf file were transferred into a huge DGL graph and saved as pickle files by the following script.

```
import os
import zipfile
import os.path as osp
from rdkit import Chem
from rdkit.Chem import ChemicalFeatures
from rdkit import RDConfig
import dgl
```

```
from dgl.data.utils import download
import torch
from collections import defaultdict
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import pathlib
import pandas as pd
import numpy as np
import pickle
_urls = {'Alchemy': 'https://alchemy.tencent.com/data/'}

# -*- coding:utf-8 -*-
"""Example dataloader of Tencent Alchemy Dataset
https://alchemy.tencent.com/
"""
import os
import zipfile
import os.path as osp
from rdkit import Chem
from rdkit.Chem import ChemicalFeatures
from rdkit import RDConfig
import dgl
from dgl.data.utils import download
import torch
from collections import defaultdict
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import pathlib
import pandas as pd
import numpy as np
_urls = {'Alchemy': 'https://alchemy.tencent.com/data/'}



fdef_name = osp.join(
"/cm/software/apps/rdkit/2019.03.02-python3.6/share/RDKit/Data",
'BaseFeatures.fdef')
chem_feature_factory = ChemicalFeatures.BuildFeatureFactory(fdef_name)

def alchemy_nodes(mol):
    """Featurization for all atoms in a molecule. The atom indices
    will be preserved.

    Args:
        mol : rdkit.Chem.rdchem.Mol
          RDKit molecule object
    Returns
        atom_feats_dict : dict
```

```
    Dictionary for atom features
"""
atom_feats_dict = defaultdict(list)
is_donor = defaultdict(int)
is_acceptor = defaultdict(int)

fdef_name = osp.join(
"/cm/software/apps/rdkit/2019.03.02-python3.6/share/RDKit/Data",
'BaseFeatures.fdef')
mol_featurizer = ChemicalFeatures.BuildFeatureFactory(fdef_name)
mol_feats = mol_featurizer.GetFeaturesForMol(mol)
mol_conformers = mol.GetConformers()
assert len(mol_conformers) == 1
geom = mol_conformers[0].GetPositions()

for i in range(len(mol_feats)):
    if mol_feats[i].GetFamily() == 'Donor':
        node_list = mol_feats[i].GetAtomIds()
        for u in node_list:
            is_donor[u] = 1
    elif mol_feats[i].GetFamily() == 'Acceptor':
        node_list = mol_feats[i].GetAtomIds()
        for u in node_list:
            is_acceptor[u] = 1

num_atoms = mol.GetNumAtoms()
for u in range(num_atoms):
    atom = mol.GetAtomWithIdx(u)
    symbol = atom.GetSymbol()
    atom_type = atom.GetAtomicNum()
    aromatic = atom.GetIsAromatic()
    hybridization = atom.GetHybridization()
    num_h = atom.GetTotalNumHs()
    atom_feats_dict['pos'].append(torch.FloatTensor(geom[u]))
    atom_feats_dict['node_type'].append(atom_type)

    h_u = []
    h_u += [
        int(symbol == x) for x in ['H', 'C', 'N', 'O', 'F', 'S', 'Cl']
    ]
    h_u.append(atom_type)
    h_u.append(is_acceptor[u])
    h_u.append(is_donor[u])
    h_u.append(int(aromatic))
    h_u += [
        int(hybridization == x)
        for x in (Chem.rdchem.HybridizationType.SP,
                  Chem.rdchem.HybridizationType.SP2,
```

```
                        Chem.rdchem.HybridizationType.SP3)
            ]
            h_u.append(num_h)
            atom_feats_dict['n_feat'].append(torch.FloatTensor(h_u))

        atom_feats_dict['n_feat'] = torch.stack(atom_feats_dict['n_feat'],
                                                dim=0)
        atom_feats_dict['pos'] = torch.stack(atom_feats_dict['pos'], dim=0)
        atom_feats_dict['node_type'] = torch.LongTensor(
            atom_feats_dict['node_type'])

        return atom_feats_dict

    def alchemy_edges(mol, self_loop=True):
        """Featurization for all bonds in a molecule. The bond indices
        will be preserved.

        Args:
          mol : rdkit.Chem.rdchem.Mol
            RDKit molecule object

        Returns
          bond_feats_dict : dict
              Dictionary for bond features
        """
        bond_feats_dict = defaultdict(list)

        mol_conformers = mol.GetConformers()
        assert len(mol_conformers) == 1
        geom = mol_conformers[0].GetPositions()

        num_atoms = mol.GetNumAtoms()
        for u in range(num_atoms):
            for v in range(num_atoms):
                if u == v and not self_loop:
                    continue

                e_uv = mol.GetBondBetweenAtoms(u, v)
                if e_uv is None:
                    bond_type = None
                else:
                    bond_type = e_uv.GetBondType()
                bond_feats_dict['e_feat'].append([
                    float(bond_type == x)
                    for x in (Chem.rdchem.BondType.SINGLE,
                              Chem.rdchem.BondType.DOUBLE,
                              Chem.rdchem.BondType.TRIPLE,
                              Chem.rdchem.BondType.AROMATIC, None)
```

```
            ])
            bond_feats_dict['distance'].append(
                np.linalg.norm(geom[u] - geom[v]))

    bond_feats_dict['e_feat'] = torch.FloatTensor(
        bond_feats_dict['e_feat'])
    bond_feats_dict['distance'] = torch.FloatTensor(
        bond_feats_dict['distance']).reshape(-1, 1)

    return bond_feats_dict

def sdf_to_dgl(sdf_file, self_loop=False):
    """
    Read sdf file and convert to dgl_graph
    Args:
        sdf_file: path of sdf file
        self_loop: Whetaher to add self loop
    Returns:
        g: DGLGraph
        l: related labels
    """
    sdf = open(str(sdf_file)).read()
    mol = Chem.MolFromMolBlock(sdf, removeHs=False)

    g = dgl.DGLGraph()

    # add nodes
    num_atoms = mol.GetNumAtoms()
    atom_feats = alchemy_nodes(mol)
    g.add_nodes(num=num_atoms, data=atom_feats)

    # add edges
    # Note here, the original model in Tencent Alchemy Git repo assumes a complete graph.
    # This leaded to low effciency when doing edge updates. Thus, some edges will be
    # deleted when I do load this graph in the next script
    if self_loop:
        g.add_edges(
            [i for i in range(num_atoms) for j in range(num_atoms)],
            [j for i in range(num_atoms) for j in range(num_atoms)])
    else:
        g.add_edges(
            [i for i in range(num_atoms) for j in range(num_atoms - 1)], [
                j for i in range(num_atoms)
                for j in range(num_atoms) if i != j
            ])

    bond_feats = alchemy_edges(mol, self_loop)
    g.edata.update(bond_feats)
```

```
    # for test set, labels are molecule ID
    l = torch.FloatTensor(target.loc[int(sdf_file.stem)].tolist()) \
        if mode in ['dev', 'valid'] else torch.LongTensor([int(sdf_file.stem)])
    return (g, l)

for mode in ["dev", "valid", "test"]: # Create the pickle file for different mode
    prop = range(12)
    transform=None
    file_dir = pathlib.Path('/home/ajp/tencent_alchemy_contest/data-bin', mode)
    if mode == 'dev':
        target_file = pathlib.Path(file_dir, "dev_target.csv")
        target = pd.read_csv(target_file,
                                index_col=0,
                                usecols=[
                                    'gdb_idx',
                                ] +
                                ['property_%d' % x for x in range(12)])
        target = target[['property_%d' % x for x in range(12)]]
    elif mode == 'valid':
        target_file = pathlib.Path(file_dir, "valid_target.csv")
        target = pd.read_csv(target_file,
                                index_col=0,
                                usecols=[
                                    'gdb_idx',
                                ] +
                                ['property_%d' % x for x in range(12)])
        target = target[['property_%d' % x for x in range(12)]]

    sdf_dir = pathlib.Path(file_dir, "sdf")
    graph_dict = {}
    graphs, labels = [], []
    for sdf_file in sdf_dir.glob("**/*.sdf"):
        result = sdf_to_dgl(sdf_file)
        graph_dict[int(sdf_file.stem)] = result[0]


    with open(mode + '_graph.pkl', 'wb') as fh:
        pickle.dump(graph_dict, fh)
```

When loading the pickle file, two major differences were made. First, all the edges which have a distance longer than the cut-off distance were deleted. This saved a huge amount of calculation time. Secondly, all Cartesian coordinates were centered at the mass center position, since property 10 is sensitive to the atom positions.

```
  # -*- coding:utf-8 -*-
"""Hacked from the example dataloader of Tencent Alchemy Dataset
https://alchemy.tencent.com/
"""
```

```
import os
import zipfile
import os.path as osp
import dgl
from dgl.data.utils import download
import torch
from collections import defaultdict
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import pathlib
from ase.data import atomic_masses
import pandas as pd
import numpy as np
import pickle
_urls = {'Alchemy': 'https://alchemy.tencent.com/data/'}


class AlchemyBatcher:
    def __init__(self, graph=None, label=None):
        self.graph = graph
        self.label = label


def batcher():
    def batcher_dev(batch):
        graphs, labels = zip(*batch)
        batch_graphs = dgl.batch(graphs)
        labels = torch.stack(labels, 0)
        return AlchemyBatcher(graph=batch_graphs, label=labels)

    return batcher_dev


def getmasscenter(g):
    '''
    Hacked from https://wiki.fysik.dtu.dk/ase/_modules/ase/atoms.html#Atoms.get_center_of_mass
    Input
    g   DGL molecular graph created by the above script
    '''
    atomic_number = g.ndata["node_type"].numpy()
    pos = g.ndata["pos"].numpy()
    m = atomic_masses[atomic_number]
    com = np.dot(m, pos) / m.sum()
    return torch.FloatTensor(com)

class PickleDataset(Dataset):

    def __init__(self, mode='dev', cutoff=5, prop = [10], transform=None):
```

```
    assert mode in ['dev', 'valid', 'test'], "mode should be dev/valid/test"
    self.mode = mode
    self.transform = transform
    self.prop = prop
    self.file_dir = pathlib.Path('/home/ajp/tencent_alchemy_contest/data-bin', mode)
    self.cut_off = cutoff

    self._load()
def cutoff(self, edges): # find those edges whcih have distance longer than cutoff
    return (edges.data['distance'] > self.cut_off).squeeze(1)

def _load(self):
    if self.mode == 'dev':
        target_file = pathlib.Path(self.file_dir, "dev_target.csv")
        self.target = pd.read_csv(target_file,
                                  index_col=0,
                                  usecols=[
                                      'gdb_idx',
                                  ] +
                                  ['property_%d' % x for x in range(12)])
        self.target = self.target[['property_%d' % x for x in self.prop]]
        pickle_dir = pathlib.Path('put your dev pickle file here')
    elif self.mode == 'valid':
        target_file = pathlib.Path(self.file_dir, "valid_target.csv")
        self.target = pd.read_csv(target_file,
                                  index_col=0,
                                  usecols=[
                                      'gdb_idx',
                                  ] +
                                  ['property_%d' % x for x in range(12)])
        self.target = self.target[['property_%d' % x for x in self.prop]]
        pickle_dir = pathlib.Path('put your valid pickle file here')
    elif self.mode =="test":
        pickle_dir = pathlib.Path('put your test pickle file here')
    self.graphs, self.labels = [], []
    with open(pickle_dir, 'rb') as fh:
        graph_data = pickle.load(fh)
    if self.mode in ["dev", "valid"]:
        for ind, idx in enumerate(self.target.index):
            g = graph_data[idx]
            central = getmasscenter(g)
            g.ndata["pos"] -= central # Shift all positions by mass center
            e_idx = g.filter_edges(self.cutoff)
            g.remove_edges(e_idx) # Remove edges which have distance longer than cut off radiu
            self.graphs.append(g)
            self.labels.append(torch.FloatTensor(self.target.loc[idx].tolist()) \
                if self.mode in ['dev', 'valid'] else torch.LongTensor([ids]))
    else:
```

```
            for idx in graph_data.keys():
                g = graph_data[idx]
                central = getmasscenter(g)
                g.ndata["pos"] -= central
                e_idx = g.filter_edges(self.cutoff)
                g.remove_edges(e_idx)
                self.graphs.append(g)
                self.labels.append(torch.IntTensor([idx]))
        self.normalize()
        print(len(self.graphs), "loaded!")

    def normalize(self, mean=None, std=None):
        labels = np.array([i.numpy() for i in self.labels])
        if mean is None:
            mean = np.mean(labels, axis=0)
        if std is None:
            std = np.std(labels, axis=0)
        self.mean = mean
        self.std = std

    def __len__(self):
        return len(self.graphs)

    def __getitem__(self, idx):
        g, l = self.graphs[idx], self.labels[idx]
        if self.transform:
            g = self.transform(g)
        return g, l
```

Finally, for Property 4, 6, 9, 10, original training/validation split were changed by adding more validation samples into training set to improve their performance in predicting big molecules.

5 Model Architecture

The architecture of SchNet can be found in Ref [4, 5, 6]. No modification was made towards SchnetPack. The architecture of SchNet wit Edge Updates is shown in Figure 1. There are two modified version compared with the standard one, one is adding a BatchNorm layer at the end of each interaction, and another one have a specific readout function towards Property 10.

The code of standard version of SchNet with Edge Updates.

```
# -*- coding:utf-8 -*-

import dgl
import torch as th
import torch.nn as nn
import numpy as np
import dgl.function as fn
from torch.nn import Softplus
```

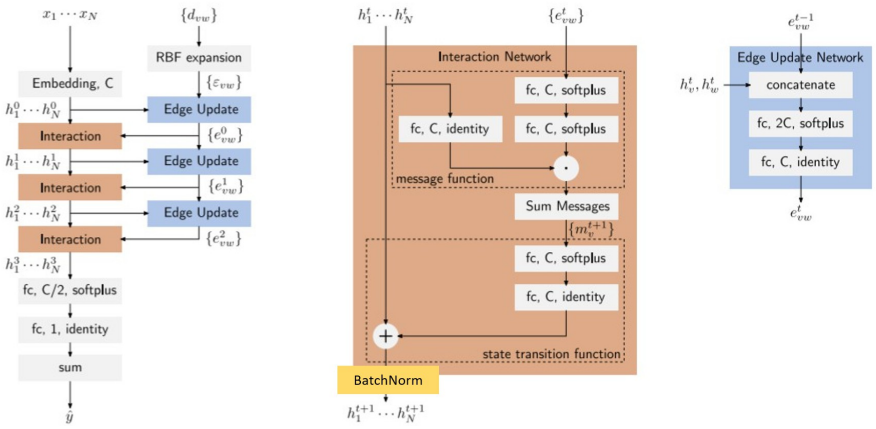

Figure 1: SchNet with Edge Updates architecture. A BatchNorm layer was added at the end of each interaction block.

```
from alchemy.layers import AtomEmbedding,  ShiftSoftplus

class RBFLayer(nn.Module):
    """
    Radial basis functions Layer.
    e(d) = exp(- gamma * ||d - mu_k||^2)
    default settings:
        gamma = 10
        0 <= mu_k <= 30 for k=1~300
    """

    def __init__(self, low=0, high=30, gap=0.1, coef=0.1, dim=1):
        super().__init__()
        self._low = low
        self._high = high
        self._gap = gap
        self._dim = dim
        self._coef = coef

        self._n_centers = int(np.ceil((high - low) / gap))
        centers = np.linspace(low, high, self._n_centers)
        self.centers = th.tensor(centers, dtype=th.float, requires_grad=False)
        self.centers = nn.Parameter(self.centers, requires_grad=False)
        self._fan_out = self._dim * self._n_centers

        self._gap = centers[1] - centers[0]

    def dis2rbf(self, edges):
        dist = edges.data["distance"]
```

```
        radial = dist - self.centers
        rbf = th.exp(-(radial**2)/self._coef)
        return {"rbf": rbf}

    def forward(self, g):
        """Convert distance scalar to rbf vector"""
        g.apply_edges(self.dis2rbf)
        return g.edata["rbf"]

class CFConv(nn.Module):
    """
    The continuous-filter convolution layer in SchNet.
    One CFConv contains one rbf layer and three linear layer
        (two of them have activation funct).
    """

    def __init__(self, rbf_dim = 64, dim=64, act="sp"):
        """
        Args:
            rbf_dim: the dimsion of the RBF layer
            dim: the dimension of linear layers
            act: activation function (default shifted softplus)
        """
        super().__init__()
        self._dim = dim
        self.linear_layer1 = nn.Linear(int(self._dim*2) + rbf_dim, int(self._dim*2))
        self.linear_layer2 = nn.Linear(int(self._dim*2), self._dim)
        self.linear_layer3 = nn.Linear(self._dim, self._dim)
        self.linear_layer4 = nn.Linear(self._dim, self._dim)

        self.activation = act

    def update_edge(self, edges):
        rbf = th.cat((edges.src['node'], edges.dst['node'], edges.data['rbf']), -1)
        rbf = self.linear_layer1(rbf)
        rbf = self.activation(rbf)
        rbf = self.linear_layer2(rbf)
        h = self.linear_layer3(rbf)
        h = self.activation(h)
        h = self.linear_layer4(h)
        h = self.activation(h)

        return {"h": h, "rbf": rbf}

    def forward(self, g):
        g.apply_edges(self.update_edge)
        g.update_all(message_func=fn.u_mul_e('new_node', 'h', 'neighbor_info'),
                     reduce_func=fn.sum('neighbor_info', 'new_node'))
```

```
        return g

class NMPEUInteraction(nn.Module):
    """
    The interaction layer in the SchNet model.
    """

    def __init__(self, rbf_dim, dim, act):
        super().__init__()
        self._node_dim = dim
        self.activation = act
        self.node_layer1 = nn.Linear(dim, dim, bias=False)
        self.cfconv = CFConv(rbf_dim, dim, act=self.activation)
        self.node_layer2 = nn.Linear(dim, dim)
        self.node_layer3 = nn.Linear(dim, dim)


    def forward(self, g):
        g.ndata["new_node"] = self.node_layer1(g.ndata["node"])
        g = self.cfconv(g)
        cf_node = g.ndata["new_node"]
        cf_node_1 = self.node_layer2(cf_node)
        cf_node_1a = self.activation(cf_node_1)
        new_node = self.node_layer3(cf_node_1a)
        g.ndata["node"] = g.ndata["node"] + new_node
        return g


class NMPEUModel(nn.Module):
    """
    NMPEU Model from:
    """

    def __init__(self,
                 dim=64,
                 cutoff=15,
                 output_dim=1,
                 width=15/150,
                 n_conv=3,
                 act = "ssp",
                 aggregation_mode="sum",
                 norm=False,
                 atom_ref=None,
                 pre_train=None):
        """
        Args:
            dim: dimension of features
            output_dim: dimension of prediction
```

```
            cutoff: radius cutoff
            width: width in the RBF function
            n_conv: number of interaction layers
            atom_ref: used as the initial value of atom embeddings,
                      or set to None with random initialization
            norm: normalization
        """
        super().__init__()
        self.name = "NMPEU"
        self._dim = dim
        self.cutoff = cutoff
        self.width = width
        self.n_conv = n_conv
        self.atom_ref = atom_ref
        self.norm = norm
        if act == "sp":
            self.activation = nn.Softplus(beta=0.5, threshold=14)
        elif act == "ssp":
            self.activation = ShiftSoftplus()
        elif act == "relu":
            self.activation = nn.ReLU()
        elif act == "selu":
            self.activation = nn.SELU()

        self.aggregation_mode = aggregation_mode

        if atom_ref is not None:
            self.e0 = AtomEmbedding(1, pre_train=atom_ref)
        if pre_train is None:
            self.embedding_layer = AtomEmbedding(dim)
        else:
            self.embedding_layer = AtomEmbedding(pre_train=pre_train)
        self.rbf_layer = RBFLayer(0, cutoff, width)
        self.conv_layers = nn.ModuleList(
            [NMPEUInteraction(self.rbf_layer._fan_out, dim, act = self.activation)]
            + [NMPEUInteraction(dim, dim, act = self.activation) for i in range(n_conv-1)])

        self.atom_dense_layer1 = nn.Linear(dim, int(dim/2))
        self.atom_dense_layer2 = nn.Linear(int(dim/2), output_dim)

    def set_mean_std(self, mean, std, device="cpu"):
        self.mean_per_atom = th.tensor(mean, device=device)
        self.std_per_atom = th.tensor(std, device=device)

    def forward(self, g):
        """g is the DGL.graph"""

        self.embedding_layer(g)
```

```
        if self.atom_ref is not None:
            self.e0(g, "e0")
        self.rbf_layer(g)# Update the edge
        for idx in range(self.n_conv):
            self.conv_layers[idx](g)

        atom = self.atom_dense_layer1(g.ndata["node"])
        atom = self.activation(atom)
        res = self.atom_dense_layer2(atom)
        g.ndata["res"] = res
        if self.atom_ref is not None:
            g.ndata["res"] = g.ndata["res"] + g.ndata["e0"]

        if self.norm:
            g.ndata["res"] = g.ndata[
                "res"] * self.std_per_atom + self.mean_per_atom
        if self.aggregation_mode == 'sum':
            res = dgl.sum_nodes(g, "res")
        elif self.aggregation_mode == "avg":
            res = dgl.mean_nodes(g, "res")
        return res
```

The BatchNorm version has additional batchnorm layer at the end of each interaction.

```
class NMPEUInteraction(nn.Module):
    """
    The interaction layer in the SchNet model.
    """

    def __init__(self, rbf_dim, dim, act):
        super().__init__()
        self._node_dim = dim
        self.activation = act
        self.node_layer1 = nn.Linear(dim, dim, bias=False)
        self.cfconv = CFConv(rbf_dim, dim, act=self.activation)
        self.node_layer2 = nn.Linear(dim, dim)
        self.node_layer3 = nn.Linear(dim, dim)
        self.bn = nn.BatchNorm1d(dim)


    def forward(self, g):
        g.ndata["new_node"] = self.node_layer1(g.ndata["node"])
        g = self.cfconv(g)
        cf_node = g.ndata["new_node"]
        cf_node_1 = self.node_layer2(cf_node)
        cf_node_1a = self.activation(cf_node_1)
        new_node = self.node_layer3(cf_node_1a)
        g.ndata["node"] = g.ndata["node"] + new_node
        g.ndata["node"] = self.bn(g.ndata["node"]) # An BatchNorm Layer is added Here
```

```
        return g
```

For Property 10 (dipole moment), a specifically designed output module is used. The idea has been used in the output module of SchNetPack), which originally come from the definition of dipole moment, which is

$$M = ||\sum_{i=0}^{n} q_i p_i|| \tag{1}$$

where q_i is the atomic charges and p_i is the cartesian positions. Besides, it is obvious that dipole moment is the norm of a vector, so all the values of property 10 need to be shifted by $min(property\ 10)$ to ensure that all values are positive.

```
class NMPEUModel(nn.Module):
    """
    NMPEU Model from:
    """

    def __init__(self,
                 dim=64,
                 cutoff=15,
                 output_dim=1,
                 width=15/150,
                 n_conv=3,
                 act = "ssp",
                 aggregation_mode="sum",
                 norm=False,
                 mean = None,
                 std = None,
                 atom_ref=None,
                 pre_train=None,
                 device = "cuda"):
        """
        Args:
            dim: dimension of features
            output_dim: dimension of prediction
            cutoff: radius cutoff
            width: width in the RBF function
            n_conv: number of interaction layers
            atom_ref: used as the initial value of atom embeddings,
                      or set to None with random initialization
            norm: normalization
        """
        super().__init__()
        self.name = "NMPEU"
        self._dim = dim
        self.cutoff = cutoff
        self.width = width
        self.n_conv = n_conv
        self.atom_ref = atom_ref
```

```
        self.norm = norm
        if act == "sp":
            self.activation = nn.Softplus(beta=0.5, threshold=14)
        elif act == "ssp":
            self.activation = ShiftSoftplus()
        elif act == "relu":
            self.activation = nn.ReLU()
        elif act == "selu":
            self.activation = nn.SELU()

        self.mean = th.tensor([mean], device = "cuda")
        self.std = th.tensor([std], device = "cuda")
        self.aggregation_mode = aggregation_mode

        if atom_ref is not None:
            self.e0 = AtomEmbedding(1, pre_train=atom_ref)
        if pre_train is None:
            self.embedding_layer = AtomEmbedding(dim)
        else:
            self.embedding_layer = AtomEmbedding(pre_train=pre_train)
        self.rbf_layer = RBFLayer(0, cutoff, width)
        self.conv_layers = nn.ModuleList(
            [NMPEUInteraction(self.rbf_layer._fan_out, dim, act = self.activation)]
            + [NMPEUInteraction(dim, dim, act = self.activation) for i in range(n_conv-1)])

        self.atom_dense_layer1 = nn.Linear(dim, int(dim/2))
        self.atom_dense_layer2 = nn.Linear(int(dim/2), output_dim)

    def forward(self, g):
        """g is the DGL.graph"""

        self.embedding_layer(g)
        if self.atom_ref is not None:
            self.e0(g, "e0")
        self.rbf_layer(g)# Update the edge
        for idx in range(self.n_conv):
            self.conv_layers[idx](g)

        pos = g.ndata["pos"]
        atom = g.ndata["node"]
        atom = self.atom_dense_layer1(atom)
        atom = self.activation(atom)
        res = self.atom_dense_layer2(atom)
        res = res * pos # atomic charges * position
        g.ndata["res"] = res
        if self.atom_ref is not None:
            g.ndata["res"] = g.ndata["res"] + g.ndata["e0"]
```

```
        if self.aggregation_mode == 'sum':
            res = dgl.sum_nodes(g, "res")
        elif self.aggregation_mode == "avg":
            res = dgl.mean_nodes(g, "res")
        res = th.norm(res, dim=1, keepdim=True) # shift values to ensure all are positive
        if self.norm:
            res = res*self.std + self.mean
        return res
```

6 Hyper-Parameter Tuning

I mainly tested three parameters, which are the hidden dimension, number of interactions and learning rate. The optimal hyperparamerters are listed for each property.

6.1 Property 0

Architecture: SchNet
Validation MAE: 3.93e-03
Parameters:

```
import schnetpack as spk
import schnetpack.representation as rep

# model build
representation = rep.SchNet(n_interactions=6, n_atom_basis=256,
                            n_filters=256, n_gaussians = 50, cutoff = 10.,
                            cutoff_network = CosineCutoff )
output_modules = [spk.atomistic.Atomwise(n_in = 256, n_out=1, property=n) for n in properties ]
model = spk.atomistic.AtomisticModel(representation, output_modules)

# build optimizer
optimizer = Adam(model.parameters(), lr=2e-4)
```

6.2 Property 1

Architecture: SchNet
Validation MAE: 4.5e-2
Code:

```
import schnetpack as spk
import schnetpack.representation as rep

# model build
representation = rep.SchNet(n_interactions=6, n_gaussians = 50,
                cutoff = 10., cutoff_network = CosineCutoff)
                output_modules = [spk.atomistic.Atomwise(n_in = 128, n_out=1,
                aggregation_mode="sum", property=n) for n in properties ]
model = spk.atomistic.AtomisticModel(representation, output_modules)
```

```
# build optimizer
optimizer = Adam(model.parameters(), lr=5e-4)
```

6.3 Property 2

I found that p2 = (1.5952847 * p9 - p4)/2.04635724 (homo = lumo - gap), so p2 is calculated by this equation with predicted p9 and p4 values.

6.4 Property 3, 5, 7, 8

Property 3, 5, 7, 8 are almost the same (MAE = 1e-5). Thus, I only trained and predicted with property 3.
Architecture: SchNet
Validation MAE: 1.16e-04
Code:

```
import schnetpack as spk
import schnetpack.representation as rep

# model build
representation = rep.SchNet(n_interactions=6, cutoff = 20)
output_modules = [spk.atomistic.Atomwise(n_in = 128, n_out=1, property=n) for n in properties ]
model = spk.atomistic.AtomisticModel(representation, output_modules)

# build optimizer
optimizer = Adam(model.parameters(), lr=5e-4)
```

6.5 Property 4

Architecture: SchNet with Edge Updates (BatchNorm version)
Validation MAE: 4.9e-2
Code:

```
from alchemy.nmpeu_batchnorm import NMPEUModel

model = NMPEUModel(dim= 128, output_dim=1, n_conv = 6,
                    act = "ssp", aggregation_mode="avg")

# build optimizer
optimizer = Adam(model.parameters(), lr=2e-4)
```

6.6 Property 6

Architecture: SchNet with Edge Updates (Standard version)
Validation MAE: 2.38e-2
Code:

```
from alchemy.nmpeu_standard import NMPEUModel
```

```
model = NMPEUModel(dim = 256, output_dim=1, act = "ssp", aggregation_mode="sum")

# build optimizer
optimizer = Adam(model.parameters(), lr=8e-5)
```

6.7 Property 9

Architecture: SchNet with Edge Updates (BatchNorm Version)
Validation MAE: 3.72e-2
Code:

```
from alchemy.nmpeu_batchnorm import NMPEUModel

model = NMPEUModel(dim = 256, output_dim=1,
                   n_conv = 8, act = "ssp", aggregation_mode="avg")

# build optimizer
optimizer = RMSprop(model.parameters(), lr=1e-4, momentum=5e-5)
```

6.8 Property 10

Architecture: SchNet with Edge Updates (dipole moment Version)
Validation MAE: 5.9e-2
Code:

```
from alchemy.nmpeu_standard_dp import NMPEUModel

model = NMPEUModel(dim= 256, output_dim=1, act = "ssp",
                   aggregation_mode="sum", norm = True,
                   mean = -1.7565487680138945, std = 1. )

# build optimizer
optimizer = Adam(model.parameters(), lr=9e-6)
```

6.9 Property 11

Architecture: SchNet
Validation MAE: 1.7e-2
Code:

```
import schnetpack as spk
import schnetpack.representation as rep

# model build
logging.info("build model")
representation = rep.SchNet(n_interactions=6, cutoff = 20)
output_modules = [spk.atomistic.Atomwise(n_in = 128, n_out=1, property=n) for n in properties ]
model = spk.atomistic.AtomisticModel(representation, output_modules)
```

```
# build optimizer
optimizer = Adam(model.parameters(), lr=1e-4)
```

7 Acknowledgement

I would like to thank my supervisors, Dr Raymond Chiong, Dr Alister Page and Dr Zhongyi Hu (Wuhan University), especially Dr Alister Page for his insightful explanation of some chemical principles. I also would like to thank Mufei Li from AWS Shanghai for his help in implementing the edge updates with DGL package.

References

[1] Albert P Bartók, Risi Kondor, and Gábor Csányi. "On representing chemical environments". In: *Phys. Rev. B* 87.18 (2013), p. 184115.

[2] Bing Huang and O Anatole von Lilienfeld. *The" DNA" of chemistry: Scalable quantum machine learning with" amons"*. arXiv preprint arXiv:1707.04146, 2017.

[3] Felix A Faber et al. "Alchemical and structural distribution based representation for universal quantum machine learning". In: *J. Chem. Phys.* 148.24 (2018), p. 241717.

[4] Kristof Schütt et al. "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions". In: *Advances in Neural Information Processing Systems*. 2017, pp. 992–1002.

[5] Kristof T Schütt et al. "SchNet–A deep learning architecture for molecules and materials". In: *The Journal of Chemical Physics* 148.24 (2018), p. 241722.

[6] KT Schütt et al. "SchNetPack: A deep learning toolbox for atomistic systems". In: *Journal of chemical theory and computation* 15.1 (2018), pp. 448–455.

[7] Peter Bjørn Jørgensen, Karsten Wedel Jacobsen, and Mikkel N Schmidt. "Neural message passing with edge updates for predicting properties of molecules and materials". In: *arXiv preprint arXiv:1806.03146* (2018).

[8] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1263–1272.

[9] Chi Chen et al. "Graph networks as a universal machine learning framework for molecules and crystals". In: *Chemistry of Materials* 31.9 (2019), pp. 3564–3572.

[10] "The atomic simulation environment—a Python library for working with atoms". en. In: 29.27 (). ISSN: 0953-8984. DOI: 10.1088/1361-648X/aa680e. URL: https://doi.org/10.1088%5C%2F1361-648x%5C%2Faa680e (visited on 04/04/2019).