```
#%%
import sampling
import decoder
import decoder_3d
from rdkit import Chem
from rdkit.Chem import Draw, AllChem
from tqdm import tqdm
import torch
from main_2dto3d_encoder_decoder_gssl import Model
#%% md

#%%
sampler = sampling.getSampler_DDIM()
z = sampler.sampling(5)
model, vb = decoder.getAE()
smiles = decoder.to_smile(model, z)
#%%
encoder_3d_instance, decoder_3d_instance = decoder_3d.getAE_3d()
#%%
mol = Chem.MolFromSmiles(smiles[0])
Draw.MolToImage(mol)
#%%
from datasets.datasets_utils import mol_to_graph_data_obj_simple_2D, mol_to_graph_data_obj_simple_3D
import torch_geometric as tgeom
from torch_geometric.data import Batch
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Geometry import Point3D
#%%
def smile_to_conformer(smiles, z, decoder_3d_instance):
    mol3d_list = []
    for idx, smi in enumerate(tqdm(smiles)):
        mol = Chem.MolFromSmiles(smi)
        mol = Chem.AddHs(mol)

        try:
            AllChem.EmbedMolecule(mol, maxAttempts=5000)
            positions = mol.GetConformers()[0].GetPositions()
        except:
            AllChem.Compute2DCoords(mol)
            positions = mol.GetConformers()[0].GetPositions()

        data = mol_to_graph_data_obj_simple_2D(mol)[0]
        data.x = data.x.float()[:, :118]
        data.edge_attr = data.edge_attr.float()
        data.n_nodes = data.x.shape[0]
        data.n_edges = data.edge_index.shape[1]
        data.pos = torch.tensor(positions).float()

        data.edge_index = tgeom.nn.radius_graph(data.pos, r=5, loop=False)
        edge_attr = torch.exp(-torch.norm(data.pos[data.edge_index[0]] - data.pos[data.edge_index[1]], dim=1))
        data.edge_attr = torch.einsum('i,j->ij', edge_attr, torch.linspace(1, 5, 16).to(edge_attr.device))
        data.n_edges = torch.tensor(data.edge_index.shape[1]).long()

        latent = z[idx].unsqueeze(dim=0).to('cuda')
        data = data.to('cuda')

        batch = Batch.from_data_list([data]).to('cuda')
```

```
        with torch.no_grad():
            pos = decoder_3d_instance(batch, latent[:, 250:])[0][-1]
        pos = pos.cpu()
        conf = mol.GetConformer()
        for jdx in range(mol.GetNumAtoms()):
            conf.SetAtomPosition(jdx, Point3D(pos[jdx, 0].item(), pos[jdx, 1].item(), pos[jdx, 2].item()))

        try:
            AllChem.MMFFOptimizeMolecule(mol)
            mol3d_list.append(mol)
        except:
            continue
    return mol3d_list

mol3d_list = smile_to_conformer(smiles, z, decoder_3d_instance)

#%%
Draw.MolsToGridImage(mol3d_list, molsPerRow=5, subImgSize=(200,200), legends=[str(x) for x in range(1, len(mol3d_list)+1)])
#%%
print(z)
#%%
from torch_geometric.nn import radius_graph
from torch_geometric.data import Data, Batch

mol_one = mol3d_list[0]
#conf_one = mol_one.GetConformer()
#positions = []
#for i in range(mol_one.GetNumAtoms()):
#    pos = conf_one.GetAtomPosition(i)
#    positions.append([pos.x, pos.y, pos.z])
data = mol_to_graph_data_obj_simple_3D(mol_one)[0]
data.x = data.x.float()[:, :118]
#data.positions = torch.tensor(positions).float()
data.edge_index = radius_graph(data.positions, r=5.0, loop=False)
#print(data.edge_index)
bt = Batch.from_data_list([data]).to('cuda')
with torch.no_grad():
        emb_3d = encoder_3d_instance(bt.x, bt.positions, bt.edge_index, bt.batch, True)[1].to('cpu')
#%%
import matplotlib.pyplot as plt
plt.plot(emb_3d[0].cpu().numpy())
plt.plot(z[0][250:].cpu().numpy())
plt.legend(['reconstructed', 'orginal'])
plt.title('3D latent space')
```