Notebook

July 28, 2024

1 Introduction

The model is a Generative Adversarial Network (GAN) that generates molecular structures in the form of SMILES (Simplified Molecular Input Line Entry System) strings. The model consists of two main components: a generator and a discriminator.

The generator takes a latent vector as input and generates a sequence of tokens representing a SMILES string. The discriminator takes a SMILES string as input and outputs a probability that the string was generated by the real data distribution rather than the generator.

The generator and discriminator are trained together using a two-step process. In the first step, the discriminator is trained to distinguish between real and fake SMILES strings. In the second step, the generator is trained to generate SMILES strings that are indistinguishable from real SMILES strings by maximizing the reward function, which is the difference between the discriminator's output for the generated string and the baseline reward.

The training process involves the following steps:

1. Sample a batch of real SMILES strings from the training data.
2. Sample a batch of latent vectors from the latent space.
3. Generate a batch of fake SMILES strings using the generator and the latent vectors.
4. Train the discriminator to distinguish between real and fake SMILES strings.
5. Train the generator to generate SMILES strings that are indistinguishable from real SMILES strings by maximizing the reward function.
6. Repeat steps 2-5 for a specified number of training steps.

After training, the generator can be used to generate new molecular structures by sampling latent vectors from the latent space and passing them through the generator. The output of the generator is a sequence of tokens representing a SMILES string, which can be converted to a molecular structure using a chemical library such as RDKit.

In the code below, the `MolGen` class is used to train the GAN on a dataset of molecular structures in SMILES format. The `generate_n` method of the `MolGen` class can be used to generate a specified number of new molecular structures, and the `Chem.MolFromSmiles` function from the RDKit library can be used to convert the generated SMILES strings to molecular structures.

2 Dataset

[18]:
```
import pandas as pd
df = pd.read_csv("/kaggle/working/molgen/qm9.csv")
```

[20]:
```
df.head()
```

[20]:
```
  mol_id smiles          A           B           C      mu  alpha    homo  \
0  gdb_1      C  157.71180  157.709970  157.706990  0.0000  13.21 -0.3877
1  gdb_2      N  293.60975  293.541110  191.393970  1.6256   9.46 -0.2570
2  gdb_3      O  799.58812  437.903860  282.945450  1.8511   6.31 -0.2928
3  gdb_4    C#C    0.00000   35.610036   35.610036  0.0000  16.28 -0.2845
4  gdb_5    C#N    0.00000   44.593883   44.593883  2.8937  12.99 -0.3604

     lumo     gap  …      zpve         u0       u298       h298       g298  \
0  0.1171  0.5048  …  0.044749 -40.478930 -40.476062 -40.475117 -40.498597
1  0.0829  0.3399  …  0.034358 -56.525887 -56.523026 -56.522082 -56.544961
2  0.0687  0.3615  …  0.021375 -76.404702 -76.401867 -76.400922 -76.422349
3  0.0506  0.3351  …  0.026841 -77.308427 -77.305527 -77.304583 -77.327429
4  0.0191  0.3796  …  0.016601 -93.411888 -93.409370 -93.408425 -93.431246

      cv     u0_atom   u298_atom   h298_atom   g298_atom
0  6.469 -395.999595 -398.643290 -401.014647 -372.471772
1  6.316 -276.861363 -278.620271 -280.399259 -259.338802
2  6.002 -213.087624 -213.974294 -215.159658 -201.407171
3  8.574 -385.501997 -387.237686 -389.016047 -365.800724
4  6.278 -301.820534 -302.906752 -304.091489 -288.720028

[5 rows x 21 columns]
```

[29]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 133885 entries, 0 to 133884
Data columns (total 21 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   mol_id     133885 non-null  object
 1   smiles     133885 non-null  object
 2   A          133885 non-null  float64
 3   B          133885 non-null  float64
 4   C          133885 non-null  float64
 5   mu         133885 non-null  float64
 6   alpha      133885 non-null  float64
 7   homo       133885 non-null  float64
 8   lumo       133885 non-null  float64
 9   gap        133885 non-null  float64
 10  r2         133885 non-null  float64
 11  zpve       133885 non-null  float64
 12  u0         133885 non-null  float64
 13  u298       133885 non-null  float64
 14  h298       133885 non-null  float64
 15  g298       133885 non-null  float64
 16  cv         133885 non-null  float64
 17  u0_atom    133885 non-null  float64
 18  u298_atom  133885 non-null  float64
 19  h298_atom  133885 non-null  float64
 20  g298_atom  133885 non-null  float64
dtypes: float64(19), object(2)
memory usage: 21.5+ MB
```

[ ]:

```python
# !pip install -r /kaggle/working/molgen/requirement.txt
```

[ ]:

```python
# !pip install --upgrade huggingface-hub==0.24.0
```

[4]:

```python
import huggingface_hub
print(huggingface_hub.__version__)
```

```
0.24.0
```

3 Loading Data and Initializing

[3]:

```python
from rdkit import Chem
from model import MolGen

# load data
data = []
with open('/kaggle/working/molgen/qm9.csv', "r") as f:
    for line in f.readlines()[1:]:
        data.append(line.split(",")[1])

# create model
gan_mol = MolGen(data, hidden_dim=64, lr=1e-3, device="cuda")
```

```
2024-07-28 15:33:12.692113: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-07-28 15:33:12.692174: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-07-28 15:33:12.693765: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
```

[17]:
```
gan_mol
```

[17]:
```
MolGen(
  (generator): Generator(
    (embedding_layer): Embedding(23, 64)
    (project): FeedForward(
      (_activations): ModuleList(
        (0): ReLU()
        (1): ELU(alpha=0.1)
      )
      (_linear_layers): ModuleList(
        (0): Linear(in_features=64, out_features=128, bias=True)
        (1): Linear(in_features=128, out_features=128, bias=True)
      )
      (_dropout): ModuleList(
        (0): Dropout(p=0.1, inplace=False)
        (1): Dropout(p=0.1, inplace=False)
      )
    )
    (rnn): LSTMCell(64, 64)
    (output_layer): Sequential(
      (0): ReLU()
      (1): Dropout(p=0.1, inplace=False)
      (2): Linear(in_features=64, out_features=128, bias=True)
      (3): ReLU()
      (4): Dropout(p=0.1, inplace=False)
      (5): Linear(in_features=128, out_features=22, bias=True)
    )
  )
  (discriminator): RecurrentDiscriminator(
    (embedding): Embedding(24, 64, padding_idx=0)
    (rnn): LstmSeq2SeqEncoder(
      (_module): LSTM(64, 64, batch_first=True, bidirectional=True)
    )
    (fc): Sequential(
      (0): ReLU()
      (1): Dropout(p=0.1, inplace=False)
      (2): Linear(in_features=128, out_features=256, bias=True)
      (3): ReLU()
      (4): Dropout(p=0.1, inplace=False)
      (5): Linear(in_features=256, out_features=1, bias=True)
      (6): Sigmoid()
    )
  )
)
```

4 Training

[5]:

```
# create dataloader
loader = gan_mol.create_dataloader(data, batch_size=128, shuffle=True,␣
 ↪num_workers=10)

# train model for 10000 steps
gan_mol.train_n_steps(loader, max_step=20000, evaluate_every=100)
```

```
/opt/conda/lib/python3.10/site-packages/torch/utils/data/dataloader.py:563:
UserWarning: This DataLoader will create 10 worker processes in total. Our
suggested max number of worker in current system is 4, which is smaller than
what this DataLoader is going to create. Please be aware that excessive worker
creation might get DataLoader running slow or even freeze, lower the worker
number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
/opt/conda/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX
is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

```
['((H5]n3]n#oNC]4=+(C', '2F4)(3=)n1=2[332oc)']
valid =  0.01
['O1OOCCO(())C(OC1OC', 'CC']
valid =  0.06
['11CC23C12', 'Cc1n#(1']
valid =  0.03
['CCCcC=OCNC', 'CCCC1CO)CcC']
valid =  0.06
['CC3C', 'C2CNO(CCNNC']
valid =  0.09
['CCCOCC1', 'CCN)C(C2Cn']
valid =  0.10
['OC(C1C3CCCN1', 'NCC1CC2OC1=C=CN3']
valid =  0.03
['CCN=CC1N1CC2C1H=O', 'CC1CNC2C112COCO13CC']
valid =  0.06
['C=OC1C(C', 'CC1CC=C=C=OC11C[']
valid =  0.04
['COCOC1', 'CCNC1CC1']
valid =  0.20
['N2C1OCC12', 'NCON#CCOC1']
valid =  0.20
```

```
/opt/conda/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX
is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

```
['OC1CcC1=CN=O', 'OC1CCC2']
valid =  0.13
['OC12C1CC2OC#N', 'OC1CC1CN1=O']
valid =  0.19
['OCC1OCCC2CC1N', 'CCC=OC1']
valid =  0.11
['OC1C2CCOC1nC1', 'CC12OC12NCC2']
valid =  0.04
['OCCC12CC#CCCOC1O', 'OCCC1C=NCC1OCC12']
valid =  0.09
['CC1CNC2OC1C2C=C2CN)', 'C=CCN1=COC1C2O']
valid =  0.10
['OC1OC2OC1C1O=O', 'O=C1CC1OCC21O']
valid =  0.13
['CC1CCCOC=CC1O', 'CC1CC2OCC1CCC=O']
valid =  0.23
['CN1=CCOC1', 'OC1CC1C21CC2OC1C=O']
valid =  0.16
['N=OCC12CC1COC12', 'C1CCOC12CC1C=N']
valid =  0.14
['C#CCOC1C1C2COC21', 'OC1CC2CC1COC2']
valid =  0.21
['O=C1CC1CNC1', 'O=COC1C=CC1CC1']
valid =  0.24
['C#CC1CC=CO1', 'CC12CCC121OC=N']
valid =  0.20
['COC1CC1CC1C', 'O=CCCCCCC1OC=N']
valid =  0.48
['O=CCC1CC1OCC1=O', 'O=OC1C1CC#CC1NC1C1']
valid =  0.26
['CCO1CCC1NC1C', 'N=C1NC1C=CN1CNC1=O']
valid =  0.40
['C3CC1CC2N3C1C2=N', 'C1C2NC=NC2C1C#C']
valid =  0.27
['OCC12CC2CC112C2', 'C1CC2NC12']
valid =  0.40
['C1OC=CN1CC=CCN=CO1', 'CC1C1OC11CCNC11CC1=']
valid =  0.37
['CC1CC2C1COC23', 'COC1CCOCC1=N']
valid =  0.48
['O=CC1OC11CCC1', 'CCC1N=NC11CC1']
valid =  0.55
['C1=CC2C=COC12', 'CN1=CC2OC12']
valid =  0.40
['CC12COC1C2N', 'CC1=COC2CC1O2']
valid =  0.55
['OCC1CCC1COC=O', 'OC1CCCOC1=O']
valid =  0.66
```

```
['OC1CN2CC1C2=N', 'OCC1CC2C1C23O']
valid =  0.57
['N=CN1CN=NC1C#N', 'CC1OC=CCCC1=O']
valid =  0.73
['CC1CC2CC12O', 'O=C1CN1C=NC1C#N']
valid =  0.65
['CCC1CC2CCNC2C1', 'CCC1OCCC1C']
valid =  0.84
['CC12CN1CC23', 'CCC#CC1OC1=O']
valid =  0.73
['C1OC=CC2CC12', 'OC1C2CC33CC1C23']
valid =  0.49
['N=CCC1CCCO1', 'CC1CC2COC12']
valid =  0.87
['CC1C2OC12', 'OCC12C3C1CC2)N=N1']
valid =  0.60
['O=C1C[C=CCOC1=O', 'OC1CC=COC1C1C=N']
valid =  0.60
['CCN1C=CC2NC12', 'CC1OCOCC=CC1']
valid =  0.80
['OCCC1C2CC3C2C3C3OC=', 'COC1OC2CCC2C1=O']
valid =  0.87
['OC1COC2COC1COC2', 'OC1CNCC11CCC1']
valid =  0.87
['CC12CC1COC2', 'CC#CCC1CNC1CO']
valid =  0.79
['O=C1CC2NC3CCC12C3=N', 'CCC12COC12C#C']
valid =  0.79
['C#CC1CC1CO', 'O=C1CN1CCC1']
valid =  0.64
['N=CC1COCC1=O', 'CCC12COC1C2']
valid =  0.79
['CC1C2C3CN=C13N=CO', 'CC#CNC1CC=CO1']
valid =  0.72
['OC1CC23CC3C1N23C', 'CC12C1NCOC2C=N']
valid =  0.72
['OC12CC3CC1CN2C1O3', 'O=C1NC=CN1CN=C=O']
valid =  0.65
['O=CN1CC2OC12', 'CC1N2C3C1CC23']
valid =  0.84
['CCOC1C2NC1C2', 'OCCC12CC1N=N2']
valid =  0.85
['C1OCC2NC12CO', 'OC#CCCCC1C2COC21']
valid =  0.79
['O=C1CC2CCC12', 'N#CC1C1NC1CC1']
valid =  0.64
['COCC1C=CC1', 'CC1C2C3OC12C3O']
valid =  0.77
```

```
['OCC12CC1CC2O', 'CC12NC1C=C2OC=O']
valid =  0.74
['CC1NC1C1CC1', 'CN1CC11CC1CO']
valid =  0.90
['CC1C2CC1CO2', 'CC12CC1COC2']
valid =  0.77
['OCC1C2CCC12', 'CC1OC2CN2C1C#N']
valid =  0.88
['CC1C2CC3C2CCOC12O3', 'CCCc1COC1C1CCCCO1']
valid =  0.70
['O=C1C=CN2CC1C=C2', 'C1OC=C2NC1CCO2']
valid =  0.87
['CCC1C2OCCC=CCO12', 'CN=CCC1OCOC1C#N']
valid =  0.87
['CC#CCC1CC1O', 'OC1CC2COC2CC1']
valid =  0.95
['CC1COC2OC2CC12', 'CCOC12CCC3C2O13']
valid =  0.83
['CC12CC1NC=CO2', 'CCCN12CC1CC=N2']
valid =  0.67
['CC#CCC1=CC2NC21', 'OC1CC23CC2C13']
valid =  0.83
['CC1C2CC3COC3N1C23', 'CCC1C2C3OC3C12']
valid =  0.87
['O=C1NC2CCC2=N1', 'O=C1C=CC2OC3CC1N23']
valid =  0.80
['OCC1CC2=NC1C2=O', 'C1CC=C=NC1C#CO']
valid =  0.75
['CCC1=CC2OC2CC1=O', 'CC#CC12CCOC1C2=O']
valid =  0.78
['CC1COC11CCCCC1', 'CC1OC2CC1C#CC2']
valid =  0.92
['O=CC1=CC=CCC2N12', 'O=C1C2CCOCC12']
valid =  0.89
['C1C1CCN=C1', 'CN1CC=CCO1']
valid =  0.72
['COCOC1CCOC1=O', 'O=CC1CC2OC1CC2=O']
valid =  0.86
['O=C1CCCN1C#N', 'CC12CC3CC1N=N23']
valid =  0.90
['CC12CNC1C1CC21', 'N#CC12NNC1C=C2']
valid =  0.87
['O=C#CC1OC2C3OC2CC13', 'O=C1NC2CC2C1C=O']
valid =  0.83
['CCC12CC1OC2', 'COCC12OC3CC1C23']
valid =  0.83
['N=C1NOC1C=O', 'CN1CN2C3OC2C1C3']
valid =  0.86
```

```
['O=C1NC2CCCC12', 'CC#CCCN1C=NC1']
valid =  0.85
['CC#CCN1CCC1=O', 'CC#CC1OCCC11CN1']
valid =  0.91
['C1C2NCC2C1c=O', 'C1OC=CC2CCN2C1=O']
valid =  0.92
['CCC12CN1C1C2C3O1', 'CCN1C2C3C=CN3C12']
valid =  0.93
['OC1C2COCOC12', 'N#CC12CC3N=C1C3ON2']
valid =  0.84
['OC1CC=CN2CC12', 'CC1C2COC1CC2=O']
valid =  0.88
['OCC1C2CC2CCC1=O', 'CN=C1NN=C1']
valid =  0.94
['CC1OC11C2CCOC12', 'COC1CN1C#N']
valid =  0.84
['C1C=CC2C3COC124NO1', 'CC1OC2CC3C2C1C3']
valid =  0.84
['OC12CC=CCOC1C2=O', 'OCC1C2C3CC2CC1C3#N']
valid =  0.92
['OC1C=CC2CC2OC1=O', 'N=C1NC=CCOC1=O']
valid =  0.90
['N=C1OC2COC2C1C#C', 'O=C1CN2CC=C12']
valid =  0.85
['N#CC1C2CC3C1C23', 'OCCC1CCOC1C=O']
valid =  0.89
['CC1OC2CN1C2=O', 'OC1C2C=CCC12']
valid =  0.91
['O=C1C2CC3CC2C1C3', 'O=CN=C1NC=C2NC1C2C#']
valid =  0.89
['COC1NC1C', 'O=C1CC2COC12']
valid =  0.91
['CCC1OC1C1CCO1', 'OC1C2COCN1C2C#N']
valid =  0.94
['C1CC2C3COC2C13', 'CCCC12OCC11CN21C']
valid =  0.91
['N=C1COC1C1CC1', 'CCC1C2CC1C1C2C11CCO']
valid =  0.86
['OC1C2C3CC2C1C3O', 'COC12CC1CCOC2']
valid =  0.96
['CC1C2NC1C1CC21', 'C1C1C11CCC23COC213']
valid =  0.91
['C1OC1C1CC2CC12', 'C1OCC2OC1C=C2']
valid =  0.93
['C=CC1CC1CC#C', 'CC1C2CC3OC2OC13']
valid =  0.91
['CC1OC2C3OC2C13', 'C#CNC1CC1C#C']
valid =  0.98
```

```
['OCC1=NC2CC2C=C1', 'O=CC1=NC2CC2C=C1']
valid =  0.91
['CCC1CC1C1CCO1', 'CC1OCOCN=NC1C=O']
valid =  0.93
['N#CCCC1HCC2CC12', 'N#CC1C2CCN12C#N']
valid =  0.96
['CC1OC2CCC2N1', 'CCC#CC1CCC1']
valid =  0.91
['C1CC2OC2CC1=O', 'CC1C2CCCC2O1']
valid =  0.98
['C1OC11C2NC3CC1CCC23', 'CC1=CCCC=C1']
valid =  0.95
['CCC#CC1CCCCC1', 'C#CC1C2OC1CC2=O']
valid =  0.92
['OC1CC=CC2=CNC12', 'C1CCC11C2CCCC12']
valid =  0.97
['O=C1C2C3C=CC1NC23', 'O=C12COC1C1COC12']
valid =  0.86
['CCC12COC1C2C#N', 'OC1C=CC2CN1C2']
valid =  0.89
['CCC1CC2OC1CO2', 'OCCC1OCC1=O']
valid =  0.96
['CC#CC1C2CC1O2', 'CC12NC1C1CCN21']
valid =  0.93
['CC12CC1COC2', 'OC1C2CC2C11CO1']
valid =  0.89
['OC1CC2CC2COC1', 'OC1CC=CC2CCN12']
valid =  0.94
['C#CC12OC1C#CC2', 'N#CCC12CC1O2']
valid =  0.93
['CCC1OC2CC1OC2', 'CC1CC23CCOC3C12']
valid =  0.95
['CC12CC1C1OC2C1C', 'COC1COC1CC#N']
valid =  0.93
['O=CCC12CC3CC1C23', 'O=C1CN2C3CC2C13C']
valid =  0.98
['O=CCC1CN=CCC=C1', 'C#CC1CC2CC2NC1']
valid =  0.97
['CCC1OC2CC1O2', 'C1C2NC3C2CC13C#N']
valid =  0.95
['N#CC1=NC2CC12', 'COC12C3NC1C2NC3']
valid =  0.95
['OCCC1NC1C#CC#N', 'COCC1C2CN1C2=O']
valid =  0.90
['CC1OCC2NC2C1=O', 'C1C=CC2NC1C2=O']
valid =  0.97
['CC1CC1OC1C1CCC1', 'OC12COC1CC21CC1']
valid =  0.92
```

```
['CC1=CC2CC3C2C13', 'CCC#CC1CCC2CC12']
valid =  0.93
['N#CC1C2NC2C1=O', 'CCC12CC3C1C2O1']
valid =  0.94
['C1CC1CCCOCO', 'CCC1C2NC1C=CC2']
valid =  0.99
['C1OCCCC11CO1', 'C#CC1C2C=CC3C1C23']
valid =  0.95
['NC1CC2CC=C12', 'CCC1CC2CCC12']
valid =  0.96
['CC1CC2CC1OC2', 'N#CC12CN3CC1C23']
valid =  0.92
['N#CC12CCC1CO2', 'CC1=CC2C3OCC3C12']
valid =  0.95
['CCCCC1OCO1', 'N=C1OCCC2CC1O2']
valid =  0.94
['CC1C=CC2OCC12', 'CCC1=C=CCC2CC1O2']
valid =  0.96
['CC1=CC2OCC2C1=O', 'O=CC1CC2NC2C1=O']
valid =  0.95
['CC12CC3OC1CCCC23', 'N#CC1NCCC1C=O']
valid =  0.95
['O=C1C=CNC2C3OC124', 'C1OC2C3C1COC23O']
valid =  0.97
['COC1C2CC3N2C13', 'O=CNC1C2CC1OC2']
valid =  0.91
['CCC1CC2CC2C11CC1', 'CC1C2OC2CC1C=O']
valid =  0.97
['C#CC12C3CC1C3C=CC2', 'O=C1CC2CCC1OC2']
valid =  0.93
['N#CC1C2C3CC2C13', 'C#CC1=CC2CCC12']
valid =  0.92
['C1OCCOC=NC=N', 'ON=C1CN2CC12']
valid =  0.95
['O=C1CN2CC1C2C#N', 'O=C1C2CC3CCOC1N23']
valid =  0.97
['OCCCN1C2CO1', 'CC1N=COCCO1']
valid =  0.95
['CC1CC11C2CCC12O', 'CN1CCOCC11CN1']
valid =  0.95
['C#CC12CC1NC2=O', 'N=C1OCCC1C#C']
valid =  0.97
['CCOC1C2C3C1C23O', 'CC1OC2CCC2C1']
valid =  0.96
['CC1COCOC1=O', 'O=CC1CC=CC2CC12']
valid =  0.92
['OC12CC3CC1COC23', 'CCCC1CCCC1C#NC']
valid =  0.95
```

```
['N#CC12CN1C1CN1CC2', 'OC1C2CC3C2CC13']
valid =  0.92
['O=CC1=NC2CN=C12', 'CC12OC1C1OCC21']
valid =  0.92
['CC12COC3C=CC1C23', 'OC12CC1CCOC2']
valid =  0.93
['CCOC1C=CC1C=O', 'N#CC1CC=CC2OC12']
valid =  0.97
['N#CC12NC3C1C3CC2O', 'O=C1C2OCC1NCC2=O']
valid =  0.95
['C1CC2OCC1OC2', 'OC1CNOC2CC12C']
valid =  0.95
['O=CCC1COCCO1', 'O=CC1NCC1CC#N']
valid =  0.97
['CC1=CC2N3CC1N23', 'C1OC11CCOC2CC12']
valid =  0.91
['COC1C2C3CC1CC23', 'O=CNC1CC1C#C']
valid =  0.95
['CC1C2C3OC=CC3C12', 'OCC1COC1CO']
valid =  0.98
['CC1C2C3OC1CC23O', 'CC1C2CCC1OC2=N']
valid =  0.97
['C#CC#CC#CC#CC1CO1', 'CC1COC2C3OC2C13']
valid =  0.91
['CCC1=CC2CCCN12', 'O=CC1NC2C3CN3C21']
valid =  0.96
['CN1C2CC3C2C3C1=O', 'OCC1CNC1C#C']
valid =  0.99
['O=CC1OCC2CC12', 'O=C1C2COC1C=C2']
valid =  0.85
['OCCCC1COC1', 'CC1NC1C1COC1']
valid =  0.98
['O=C1CCNC2CCC12', 'OC1CN=CC2NC12']
valid =  0.99
['O=CC1CCOC1=NO', 'COCC1C2CC1OC2']
valid =  0.94
['OC12CCC=CC1C2', 'OC1C2CC3C1C=CC23']
valid =  0.97
['CN1C=CC=CCC1=O', 'C#CC1C2C3COC1N23']
valid =  0.97
['OCCCC1OC2C1C2', 'OCC12COC1CC2=O']
valid =  0.91
['O=CCC1CC2CCC12', 'O=CNC1CCCC1']
valid =  0.95
['O=CCNCC1CC1', 'CC12CN3CC1C2CO3']
valid =  0.91
['CCN1C2COC12C=O', 'O=C1C2C=CC3N3C1C2']
valid =  0.96
```

```
['OC12CC3C12OC3C1CC1', 'CC12CN1CC1OC2O1']
valid =  0.85
['C#CC12COC=NC12C', 'CC1CN1C12CCC1C1OC21']
valid =  0.92
['N#CC1CN1CC=O', 'CN1C2CC2C11CC1']
valid =  0.96
['CCC1OCC1C1CC1', 'CCOCC1C2CC12']
valid =  0.97
['C1C2COC=NC2C1=O', 'CC1CCCC=CC1=O']
valid =  0.99
['C1CC2=C=CCN1C2=O', 'O=CC1CC2OC2C1=O']
valid =  0.96
['C1C2OC3CC3CN2C1', 'OC1CC2N1C2C#C']
valid =  0.92
['O=C1C2OC11CNC2C1', 'CC12OC1C1OCC12O']
valid =  0.97
['CN=C1COC1C#N', 'CN=C1OC2C3CC1N23']
valid =  0.90
['C1C2CC3CC2C31', 'N#CC1CC=CC2CCC12']
valid =  0.98
['C1CN1C1C2CCC=C1O2', 'OCC12CC1CC1CN21']
valid =  0.96
['C#CC1C2NC2C11CC1', 'OCC12C3OC1CCC23']
valid =  0.92
['C1CC2N3CC2C3C1', 'N#CC1C2C3C=CC1C23']
valid =  0.97
['CC12OCC1NC2=N', 'OC12CCC1N1CC21']
valid =  0.99
['O=CNC1CC11CN1', 'N#CC1=NCC2CC12']
valid =  0.97
['O=C1OC2CC1NC2=O', 'CC1CC1C1OC2CC12']
valid =  0.96
['CCCCCN1CC2CC12', 'O=C1C2C=C=CC1COC2']
valid =  0.97
['OC1COCC=CC1=O', 'COC1C2CCC12C#C']
valid =  0.99
['CC12OCC3CN1C23C#', 'CC1CCCC2CC1C2']
valid =  0.98
['CC1COCC2OC12C#N', 'COCC12CC3CC1C23']
valid =  0.98
['CN1CC1OCC=O', 'C1CC23NC1=CN=N1C231']
valid =  0.93
```

[6]:
```
gan_mol.eval()

print('ok')
```

```
ok
```

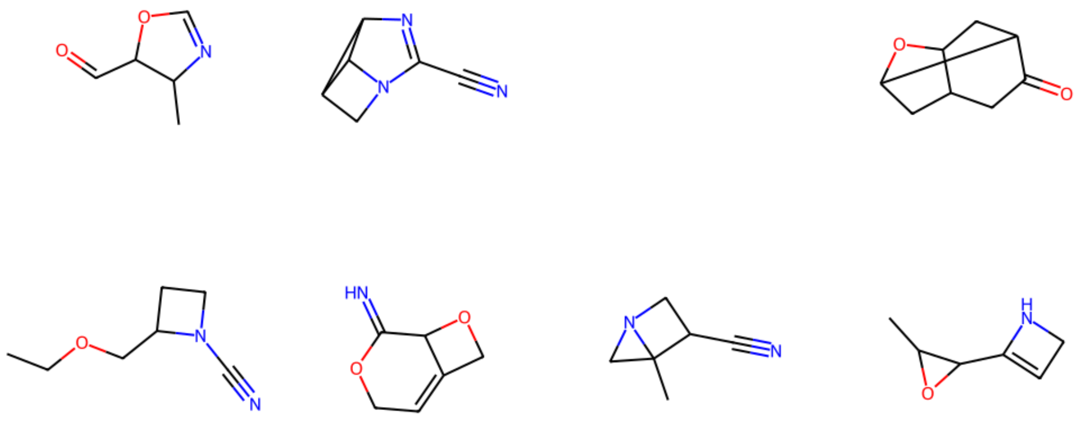
5 Generating Smiles

[7]:
```
# After training
# generate Smiles molecules
smiles_list = gan_mol.generate_n(8)

# convert with rdkit
mol_list = [Chem.MolFromSmiles(m) for m in smiles_list]

# draw
Chem.Draw.MolsToGridImage(mol_list, molsPerRow=4, subImgSize=(250, 250),␣
 ↪maxMols=10)
```

[7]:



[14]:
```
smiles_list
```

[14]:
```
['CC1N=COC1C=O',
 'N#CC1=NC2C3CN1C23',
 'O=CCCC#CC11CO1',
 'O=C1CC2CC3OC2CC13',
 'CCOCC1CCN1C#N',
 'N=C1OCC=C2COC12',
 'CC12CN1CC2C#N',
 'CC1OC1C1=CCN1']
```

[21]:
```
mol_list
```

[21]:
```
[<rdkit.Chem.rdchem.Mol at 0x7ef8e301a8f0>,
 <rdkit.Chem.rdchem.Mol at 0x7ef8e301ac70>,
 None,
 <rdkit.Chem.rdchem.Mol at 0x7ef8e301aab0>,
 <rdkit.Chem.rdchem.Mol at 0x7ef8e301a650>,
 <rdkit.Chem.rdchem.Mol at 0x7ef8e301a490>,
 <rdkit.Chem.rdchem.Mol at 0x7ef8e301a340>,
 <rdkit.Chem.rdchem.Mol at 0x7ef8e301ace0>]
```

The none in `mol_list` indicates that it is not valid.

[ ]:

```
# dir(gan_mol)
```

6 Saving and loading model

[11]:

```
# Save model
import torch
torch.save(gan_mol.state_dict(), 'gan_mol_dict.pth')
```

[24]:

```
# Load model
gan_mol_n = MolGen(data, hidden_dim=64, lr=1e-3, device="cuda")

# Load the state dictionary into the new model
gan_mol_n.load_state_dict(torch.load('gan_mol_dict.pth'))

# Print the loaded model state to verify
# print("Loaded model state:", gan_mol_n.state_dict())
```

[24]:

```
<All keys matched successfully>
```

[27]:

```
gan_mol_n.eval()

print('ok')
```

```
ok
```

[28]:

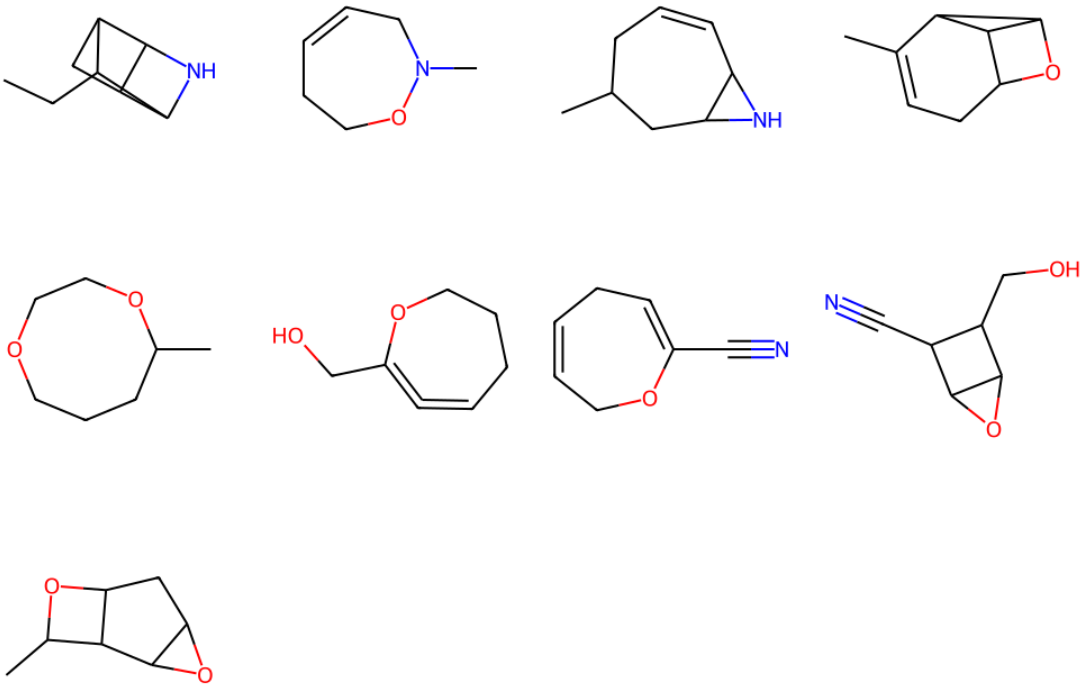
```
# After training
# generate Smiles molecules
smiles_list = gan_mol_n.generate_n(12)

# convert with rdkit
mol_list = [Chem.MolFromSmiles(m) for m in smiles_list]

# draw
Chem.Draw.MolsToGridImage(mol_list, molsPerRow=4, subImgSize=(250, 250),␣
 ↪maxMols=10)
```

```
/opt/conda/lib/python3.10/site-packages/rdkit/Chem/Draw/IPythonConsole.py:261:
UserWarning: Truncating the list of molecules to be displayed to 10. Change the
maxMols value to display more.
  warnings.warn(
```

[28]:


NH
N
O
NH
O
O
O
HO
O
O
N
N
OH
O
O
O