ZINC_DB_molecule_generation_using_LSTM_AutoEncoder

January 13, 2022

Files

smiles : Virtual Environment - Use source /smiles/bin/activate to activate the virtual env datasets/download.sh : Bash script to download datasets Usage : bash download.sh gdb13 OR bash download.sh gdb17 or bash download.sh gdb17 datasets/delete.sh : bash script to delete files in the folder. Usage : bash delete.sh gdb11_ where gdb11 is the grep string to search for. datasets/extract.sh : To extract files from a tar file and to save it in a folder

1 About the dataset

The dataset was downloaded from https://zinc.docking.org/substances/subsets/for-sale.csv?count=all with 'for_sale' filter

2 Functions

[38]:
```
# Function to One hot encode the data into into 2 Dimensions.
# Splitting the dataset to len(distinct_charachter) number of columns
def OneHotEncoding(data,ohe_dict,max):
  """ The function makes use of the ohe_helper function to One Hot Encode a␣
 ↪list of charachters and to return
  the encoded format. Since the input here is a string, the output will be of 2␣
 ↪Dimensions """

  import numpy as np
  ohe=[]
  for data_point in data:
    try:
      ohe.append(ohe_helper(data_point,ohe_dict,max))
    except:
      print(data_point)
  return ohe

def ohe_helper(data,ohe_dict,max_value):
  """ Helper function to One Hot Encode the data.
  It is used by the above function"""

  import numpy as np
```

```
  out=np.zeros((max_value,len(ohe_dict)))
  for i,element in enumerate(data):
    out[i][key_value[element]]=1

  return out

def save(file,key_values):
  """ The function will take the data and save it as an npz file, with
  name smiles.npz"""
  import numpy as np
  np.savez('smiles.npz',ohe=temp,key=key_values)

def OneHotDecoding_helper(data,dictionary):
  """ Function to decode a One Hot Encoded 2D data back into its smile␣
 ↪representation"""
  key=list(dictionary.keys())
  string=''
  for charachter in data:
    idx=charachter.argmax()
    string+=key[idx]
  return string

def check_smiles(string):
  object=check_molecule(string)
```

3 Define auto encoder

[2]:
```
def print_distributions(number_of_neurons, data):
  import matplotlib.pyplot as plt
  import seaborn as sns

  plt.figure(figsize=(20,20))

  for i in range(number_of_neurons):
    plt.subplot(8,number_of_neurons//8,i+1)
    sns.histplot(data[i])

def print_decoder_outputs(input,predictions,number):
  import matplotlib.pyplot as plt
  plt.figure(figsize=(10,3))
  for i in range(number):
    plt.subplot(1,number,i+1)
    plt.axis='off'
    plt.imshow(input[i].reshape(28,28),cmap='gray')
  plt.figure(figsize=(10,3))
  for i in range(number):
```

```
    plt.subplot(1,number,i+1)
    plt.axis='off'
    plt.imshow(predictions[i].reshape(28,28),cmap='gray')
```

4 Preparing the data

[3]:
```
# Connecting to google drive and navigating to folder
from google.colab import drive
drive.mount('/content/drive')
import os
os.chdir("drive/MyDrive/Colab Notebooks/molecule_generation_SMILES/scripts")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[4]:
```
!pip install smilite
# Importing Libraries
import pandas as pd
import smilite # Library to check if data generated is correct or not
from check_molecule import *
import numpy as np
```

```
Requirement already satisfied: smilite in /usr/local/lib/python3.7/dist-packages
(2.3.0)
Requirement already satisfied: PyPrind>=2.3.1 in /usr/local/lib/python3.7/dist-
packages (from smilite) (2.11.3)
```

[26]:
```
data=pd.read_csv('ZINC.csv',nrows=7000000)
```

[27]:
```
data.head()
```

[27]:

	zinc_id	smiles
0	ZINC000000000007	C=CCc1ccc(OCC(=O)N(CC)CC)c(OC)c1
1	ZINC000000000010	C[C@@]1(c2ccccc2)OC(C(=O)O)=CC1=O
2	ZINC000000000011	COc1cc(Cc2cnc(N)nc2N)cc(OC)c1N(C)C
3	ZINC000000000012	O=C(C[S@@](=O)C(c1ccccc1)c1ccccc1)NO
4	ZINC000000000014	CC[C@H]1[C@H](O)N2[C@H]3C[C@@]45c6ccccc6N(C)[C…

[28]:
```
zinc_id_from_db=data['zinc_id']
data.drop(['zinc_id'],axis=1,inplace=True)
data.columns=['Molecule']


data.head()
```

[28]:

	Molecule
0	C=CCc1ccc(OCC(=O)N(CC)CC)c(OC)c1
1	C[C@@]1(c2ccccc2)OC(C(=O)O)=CC1=O

```
2                  COc1cc(Cc2cnc(N)nc2N)cc(OC)c1N(C)C
3               O=C(C[S@@](=O)C(c1ccccc1)c1ccccc1)NO
4  CC[C@H]1[C@H](O)N2[C@H]3C[C@@]45c6ccccc6N(C)[C…
```

One Hot Encoding the data

[29]:
```
# % time


# Creating a new column with the charachter seperated as a list
data['Molecule_sep']=data['Molecule'].apply(lambda x: list(x))

#Creating a new column with length of each smiles representation
data['length']=data['Molecule_sep'].apply(lambda x: len(x))
```

[31]:
```
# Finding the distinct charachters in the dataset
distinct_charachters=set(' '.join(data[data['length']==32]['Molecule'].values))

# Creating a dictionary to get the index value
key_value={}
key_value[' ']=0
idx=1
for charachter in distinct_charachters:
  if charachter!=' ':
    key_value[charachter]=idx
    idx+=1

data=data[data['length']==32] # Filtering the data to get elements of same size
data.to_csv('zinc_data_reduced.csv')
np.savez('key_value_pairs',key=key_value)
```

[91]:
```
data=pd.read_csv('zinc_data_reduced.csv')
data.drop(data.columns[0],axis=1,inplace=True)
key_value=np.load('key_value_pairs.npz',allow_pickle=True)
key_value=key_value['key']
key_value=key_value.item()
```

[64]:
```
#Since the charachters are not ordinal , converting them into one Hot Encoding
import numpy as np
temp=np.array(OneHotEncoding(data['Molecule'],key_value,data['length'].max()))

np.savez('smiles_zinc.npz',ohe=temp,key=key_value)
```

[65]:
```
# Reading the saved Nump file
import numpy as np
data_en=np.load('smiles_zinc.npz',allow_pickle=True)
key_value=data_en['key'].item()
data_en=data_en['ohe']
```

5 LSTM Autoencoder

Reference : TowardsDataScience post on LSTM AutoEncoders by Chitta Ranjan

[67]:
```
OneHotDecoding_helper(data_en[0],key_value)
```

[67]:
```
'C=CCc1ccc(OCC(=O)N(CC)CC)c(OC)c1'
```

[68]:
```
slicing_point=int(len(data_en)*0.75)
x_train=data_en[:slicing_point]
x_test=data_en[slicing_point:]
```

[ ]:
```
# The motive here is to create a Deep autoencoder
from keras import Sequential
from keras.layers import LSTM,Dense,RepeatVector
# define model
model = Sequential()
model.add(LSTM(32, activation='relu', input_shape=(32,32),␣
 ↪return_sequences=True))
model.add(LSTM(8, activation='relu', return_sequences=False))
model.add(RepeatVector(32))
model.add(LSTM(64, activation='relu', return_sequences=True))
model.add(LSTM(128, activation='relu', return_sequences=True))
model.add(Dense(32, activation='softmax'))
model.compile(optimizer='adam', loss='binary_crossentropy')
model.summary()

model.fit(x_train,x_train,epochs = 30,
          batch_size=512,
          shuffle = True,
          validation_data = (x_test,x_test))
```

```
WARNING:tensorflow:Layer lstm_12 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.
WARNING:tensorflow:Layer lstm_13 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.
WARNING:tensorflow:Layer lstm_14 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it doesn't
meet the criteria. It will use a generic GPU kernel as fallback when running on
GPU.
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

```
 lstm_12 (LSTM)              (None, 32, 32)            8320

 lstm_13 (LSTM)              (None, 8)                 1312

 repeat_vector_3 (RepeatVect  (None, 32, 8)            0
 or)

 lstm_14 (LSTM)              (None, 32, 64)            18688

 lstm_15 (LSTM)              (None, 32, 128)           98816

 dense_3 (Dense)             (None, 32, 32)            4128

=================================================================
Total params: 131,264
Trainable params: 131,264
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
124/124 [==============================] - 39s 280ms/step - loss: 0.2334 -
val_loss: 0.0981
Epoch 2/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0948 -
val_loss: 0.0922
Epoch 3/30
124/124 [==============================] - 34s 271ms/step - loss: 0.0896 -
val_loss: 0.0868
Epoch 4/30
124/124 [==============================] - 34s 271ms/step - loss: 0.0863 -
val_loss: 0.0848
Epoch 5/30
124/124 [==============================] - 34s 271ms/step - loss: 0.0837 -
val_loss: 0.0834
Epoch 6/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0818 -
val_loss: 0.0809
Epoch 7/30
124/124 [==============================] - 34s 273ms/step - loss: 0.0797 -
val_loss: 0.0794
Epoch 8/30
124/124 [==============================] - 34s 272ms/step - loss: 0.0781 -
val_loss: 0.0773
Epoch 9/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0765 -
val_loss: 0.0768
Epoch 10/30
124/124 [==============================] - 34s 272ms/step - loss: 0.0743 -
val_loss: 0.0722
```

```
Epoch 11/30
124/124 [==============================] - 34s 271ms/step - loss: 0.0738 -
val_loss: 0.0775
Epoch 12/30
124/124 [==============================] - 34s 270ms/step - loss: 0.0713 -
val_loss: 0.0727
Epoch 13/30
124/124 [==============================] - 33s 268ms/step - loss: 0.0711 -
val_loss: 0.0691
Epoch 14/30
124/124 [==============================] - 33s 269ms/step - loss: 0.0704 -
val_loss: 0.0727
Epoch 15/30
124/124 [==============================] - 33s 269ms/step - loss: 0.0714 -
val_loss: 0.0714
Epoch 16/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0692 -
val_loss: 0.0712
Epoch 17/30
124/124 [==============================] - 34s 271ms/step - loss: 0.0672 -
val_loss: 0.0681
Epoch 18/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0705 -
val_loss: 0.0679
Epoch 19/30
124/124 [==============================] - 34s 271ms/step - loss: 0.0666 -
val_loss: 0.0655
Epoch 20/30
124/124 [==============================] - 33s 269ms/step - loss: 0.0663 -
val_loss: 0.0737
Epoch 21/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0658 -
val_loss: 0.0644
Epoch 22/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0635 -
val_loss: 0.0643
Epoch 23/30
124/124 [==============================] - 33s 270ms/step - loss: 0.0688 -
val_loss: 0.0659
Epoch 24/30
124/124 [==============================] - 33s 268ms/step - loss: 0.0652 -
val_loss: 0.0638
Epoch 25/30
124/124 [==============================] - 34s 273ms/step - loss: 0.0629 -
val_loss: 0.0653
Epoch 26/30
124/124 [==============================] - 34s 272ms/step - loss: 0.0648 -
val_loss: 0.0633
```

```
Epoch 27/30
 78/124 [==================>...] - ETA: 11s - loss: 0.0614
```

[73]:
```
# Seperating the encoder and the decoder
from keras.models import Model
encoder=Model(inputs=model.input,outputs=model.layers[2].output)
decoder=Model(inputs=model.layers[-3].input,outputs=model.output)
```

[74]:
```
#Getting the encoded format of the data
encoded_data=encoder.predict(data_en)

encoded_data.shape
```

[74]:
```
(84044, 32, 8)
```

[75]:
```
gaussian=[] # Storing the outputs of individial neurons to different lists
for data in encoded_data:
  neuron_data=[]
  for layer in data: # 18 layers
    for value in layer : # Each layer has 2 values
      neuron_data.append(value)
  gaussian.append(neuron_data)
gaussian=np.array(gaussian)
```

[77]:
```
hist=[[] for i in range(256)]
for data in gaussian:
  for i,value in enumerate(data):
    hist[i].append(value)
```

[78]:
```
smiles=decoder.predict(encoded_data)
```

[79]:
```
len(smiles)
```

[79]:
```
84044
```

[81]:
```
t=smilite.get_zincid_from_smile('Cc1ccc(CC==))==O)c2ccc(C)cc22)c1')
t
```

```
Invalid SMILE string Cc1ccc(CC==))==O)c2ccc(C)cc22)c1
```

[81]:
```
[]
```

[82]:
```
molecules
```

[82]:
```
['CCCCCCCCCCCCCCCCCCCC(=O)c1ccccc1',
 'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC',
 'CCCCCCCCCCCCCCCCCCCCCCCCCCCC(CO)O']
```

[83]:
```
i=0
for molecule in smiles:
  smiles_string=OneHotDecoding_helper(molecule,key_value)
  if len(smilite.get_zincid_from_smile(smiles_string))>0:
    molecules.append(smiles_string)
    print(smiles_string)
  if i%50==0:
    print(i+1,' generations scanned')
  if len(molecules)>3:
    break
  i+=1
```

```
1  generations scanned
Invalid SMILE string CC(C)O))c1ccc(CC@@H](CCCCC=O)cc1
Invalid SMILE string CCCCCCCCCCC==))c1ccccccccc2ccc1
Invalid SMILE string COc1ccc(CC==))CCCCCCCC(())OO)cc1
Invalid SMILE string CCCCCCCCCCCCCC(=C)c1ccccc1))(C)O
Invalid SMILE string CCC(==))c1ccccc1CCCCCCCCC))((C)C
Invalid SMILE string Cc1ccc(CC==@))CCCCCCC))Cc11cccc1
Invalid SMILE string CCCCCCCCCCC==))c1ccccccccccc221
Invalid SMILE string CCC(=O)cc1ncccccccccc1))CCCCCC2
Invalid SMILE string CC(CO))c1ccc(CC=O))c2ccccc2F)cc1
Invalid SMILE string Cc1ccc(Ccccccccccc(CCCCCC)))2))c1
Invalid SMILE string C=CCCCCCCC==))cc1ccccc1)c1ccccc1
Invalid SMILE string C=CCCCCCC=)))c1cccccc11)c1ccccc1
Invalid SMILE string CCCCCCCCC==))c1cccc(cccc1)CCCCC1
Invalid SMILE string CCCCCCCCCCC==)CC==O)c1ccc(Cl)cc1
Invalid SMILE string CCCCCCCCCC==)))c1ccc((CC((CCC)c1
Invalid SMILE string C=C(CCCCC=)))c1cccccc11)c1cccc21
Invalid SMILE string C=CCCCCCC==))c1cccccc11)c1ccccc1
Invalid SMILE string Cc1ccc(CC(=O)Cc2ccccc22CC)))))c1
Invalid SMILE string CCCCCCCCCCCC==)))c1cc(CCC))O)cc1
Invalid SMILE string CCCC==))c1ccc(Occ2ccccccc2cccc21
Invalid SMILE string CCCCCCCCC==)))c1ccccccccccc2)cc1
Invalid SMILE string Cc1ccc(c)cc1CC==))c1ccccc1CC(C))
Invalid SMILE string Cc1ccc(ccccccCCCC=))CCCCCC))nH]1
Invalid SMILE string Cc1ccc(Ccccccccccccc CCCCC))C)c12
51  generations scanned
Invalid SMILE string C=CCCCCCCC==))c1ccccc1)c1ccccc12
Invalid SMILE string COc1ccc(CCcccCCCCCC))))Cc1cccc12
Invalid SMILE string Cc1ccc(ccccc1CCC==))CCCCCC))nH]1
Invalid SMILE string CCCC===CCc1ccccccccc()))c22)c12
c1ccc(CCCCCCCCCCCCCCc2ccccc2)cc1
```

[84]:
```
print('The successful smiles string generated are ')
molecules
```

```
The successful smiles string generated are
```

[84]:
```
['CCCCCCCCCCCCCCCCCCCC(=O)c1ccccc1',
 'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC',
 'CCCCCCCCCCCCCCCCCCCCCCCCCCCC(CO)O',
 'c1ccc(CCCCCCCCCCCCCCc2ccccc2)cc1']
```

6 Checking if the molecules generated are present in the dataset or if they are new

[92]:
```
for molecule in molecules:
  if len(data[data['Molecule']==molecule].index.values)==0:
    print(molecule,'  not found in dataset')
  else:
    print(molecule,' found in dataset')
```

```
CCCCCCCCCCCCCCCCCCCC(=O)c1ccccc1   not found in dataset
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC  found in dataset
CCCCCCCCCCCCCCCCCCCCCCCCCCCC(CO)O   not found in dataset
c1ccc(CCCCCCCCCCCCCCc2ccccc2)cc1   not found in dataset
```

[93]:
```
# Getting zinc ID of the generated molecules
zinc_id=[]
for molecule in molecules:
  zinc_id.append(smilite.get_zincid_from_smile(molecule))
```

[94]:
```
import pandas as pd
new_molecules=pd.DataFrame()
new_molecules['Generated Molecules']=molecules
new_molecules['Zinc_ID']=zinc_id
new_molecules
```

[94]:

	Generated Molecules	Zinc_ID
0	CCCCCCCCCCCCCCCCCCCC(=O)c1ccccc1	[ZINC000115464572]
1	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC	[ZINC000006920423]
2	CCCCCCCCCCCCCCCCCCCCCCCCCCCC(CO)O	[ZINC000103820528, ZINC000103820533]
3	c1ccc(CCCCCCCCCCCCCCc2ccccc2)cc1	[ZINC000140966601]

7 Saving the models and data

[95]:
```
import pickle
pickle.dump(model,open('LSTM_VAE_zinc.sav','wb'))
pickle.dump(encoder,open('Encoder_zinc.sav','wb'))
pickle.dump(decoder,open('Decoder_zinc.sav','wb'))
pickle.dump(data_en,open('encoded_input_zinc.dat','wb'))
pickle.dump(encoded_data,open('encoder_output_zinc.dat','wb'))
pickle.dump(smiles,open('decoder_output_zinc.dat','wb'))
```

```
INFO:tensorflow:Assets written to:
ram://d7362761-824c-4669-8b89-0c38cb2662c1/assets

WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc82e1a1d50> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc8167cbb10> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc8167e7750> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc8167a2190> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.

INFO:tensorflow:Assets written to:
ram://ce8780c4-f31c-4a66-b1b1-7b41e9b8c3bb/assets

INFO:tensorflow:Assets written to:
ram://ce8780c4-f31c-4a66-b1b1-7b41e9b8c3bb/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc82e1a1d50> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc8167cbb10> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.

INFO:tensorflow:Assets written to:
ram://2d6f00e0-b4f3-4a3b-b40f-979dec50cbee/assets

INFO:tensorflow:Assets written to:
ram://2d6f00e0-b4f3-4a3b-b40f-979dec50cbee/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc8167e7750> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7fc8167a2190> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
```

8 Working with latent space to generate new molecules

[107]:
```
smiles_predictions=[OneHotDecoding_helper(smile,key_value) for smile in smiles]
```

[179]:
```
index_list_of_valid_zinc15_predictions=[]
encoder_values=[]
for prediction in molecules:
  idx=(smiles_predictions.index(prediction))
  index_list_of_valid_zinc15_predictions.append(idx)
  encoder_values.append(encoded_data[idx])
encoder_values=np.array(encoder_values)
```

[217]:
```
for pos in range(8):
  print('Value of pos : ',pos)
  print(encoder_values[0][0][pos])
  print(encoder_values[1][0][pos])
  print(encoder_values[2][0][pos])
  print(encoder_values[3][0][pos])
```

```
Value of pos :  0
0.0
0.0
0.0
0.0
Value of pos :  1
0.0
0.0
0.0
```

```
0.0
Value of pos :  2
3.117608
2.7760482
2.4487445
8.36374
Value of pos :  3
4.28397
3.5332496
3.6849833
2.0634527
Value of pos :  4
3.346471
2.9913094
2.9494824
17.87391
Value of pos :  5
2.7407904
1.4493
1.5021274
2.0587478
Value of pos :  6
9.601776
11.649847
11.464967
20.242287
Value of pos :  7
2.3084958
2.6623216
2.4970162
24.374287
```

[236]:
```
neuron0=0
neuron1=0
neuron2=0
neuron3=0
neuron4=0
neuron5=0
neuron6=0
neuron7=0
manual_encoding=[neuron0,neuron1,neuron2,neuron3,neuron4,neuron5,neuron6,neuron7]
manual_encoding=[manual_encoding for i in range(32)]
manual_encoding=np.array(manual_encoding)
temp=decoder.predict(np.expand_dims(manual_encoding,axis=0))
smiles_generated=OneHotDecoding_helper(temp[0],key_value)
print(smiles_generated)
if len(smilite.get_zincid_from_smile(smiles_generated))>0:
```

```
  print(smilite.get_zincid_from_smile(smiles_generated))
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
['ZINC000006920423']
```

8.1 If all neurons are 0, it produces all Carbon atoms

[248]:
```
neuron0=0
neuron1=0
neuron2=np.random.uniform(0,8)
neuron3=0
neuron4=0
neuron5=0
neuron6=0
neuron7=0
manual_encoding=[neuron0,neuron1,neuron2,neuron3,neuron4,neuron5,neuron6,neuron7]
manual_encoding=[manual_encoding for i in range(32)]
manual_encoding=np.array(manual_encoding)
temp=decoder.predict(np.expand_dims(manual_encoding,axis=0))
smiles_generated=OneHotDecoding_helper(temp[0],key_value)
print(smiles_generated)
if len(smilite.get_zincid_from_smile(smiles_generated))>0:
  print(smilite.get_zincid_from_smile(smiles_generated))
```

```
OOOCCCCC111111ccccccc1111=(1)c11
```

8.2 Neuron 2 adds 0, 1 and c to the Molecule

[252]:
```
neuron0=0
neuron1=0
neuron2=np.random.uniform(0,8)
neuron3=np.random.uniform(2.44,4.28)
neuron4=0
neuron5=0
neuron6=0
neuron7=0
manual_encoding=[neuron0,neuron1,neuron2,neuron3,neuron4,neuron5,neuron6,neuron7]
manual_encoding=[manual_encoding for i in range(32)]
manual_encoding=np.array(manual_encoding)
temp=decoder.predict(np.expand_dims(manual_encoding,axis=0))
smiles_generated=OneHotDecoding_helpaer(temp[0],key_value)
print(smiles_generated)
if len(smilite.get_zincid_from_smile(smiles_generated))>0:
  print(smilite.get_zincid_from_smile(smiles_generated))
```

```
OOCCCCCCCCCCCCCCC--111111111111++
```

8.3 Neuron 3 adds + and - to the Molecule. So it works when other combinatons are also in place

[257]:
```
neuron0=0
neuron1=0
neuron2=np.random.uniform(0,8)
neuron3=0
neuron4=np.random.uniform(2.9,17.87)
neuron5=0
neuron6=0
neuron7=0
manual_encoding=[neuron0,neuron1,neuron2,neuron3,neuron4,neuron5,neuron6,neuron7]
manual_encoding=[manual_encoding for i in range(32)]
manual_encoding=np.array(manual_encoding)
temp=decoder.predict(np.expand_dims(manual_encoding,axis=0))
smiles_generated=OneHotDecoding_helper(temp[0],key_value)
print(smiles_generated)
if len(smilite.get_zincid_from_smile(smiles_generated))>0:
  print(smilite.get_zincid_from_smile(smiles_generated))
```

```
OCCCOOOCc1cccccc1111c11ccccN1o4o
```

8.4 Neuron 4 adds N and o to molecules

[266]:
```
neuron0=0
neuron1=0
neuron2=np.random.uniform(0,8)
neuron3=0
neuron4=np.random.uniform(2.9,17.87)
neuron5=np.random.uniform(1.4,2.7)
neuron6=0
neuron7=0
manual_encoding=[neuron0,neuron1,neuron2,neuron3,neuron4,neuron5,neuron6,neuron7]
manual_encoding=[manual_encoding for i in range(32)]
manual_encoding=np.array(manual_encoding)
temp=decoder.predict(np.expand_dims(manual_encoding,axis=0))
smiles_generated=OneHotDecoding_helper(temp[0],key_value)
print(smiles_generated)
if len(smilite.get_zincid_from_smile(smiles_generated))>0:
  print(smilite.get_zincid_from_smile(smiles_generated))
```

```
CCCCCCccccccNO1N1[CC[[CH[ooC5C52
```

8.5 Neuron 5 adds [and 5 to molecule. Hence restricting its usage

[273]:
```
neuron0=0
neuron1=0
neuron2=np.random.uniform(0,8)
```

```
neuron3=0
neuron4=np.random.uniform(2.9,17.87)
neuron5=0
neuron6=0
neuron7=np.random.uniform(2.3,24.37)
manual_encoding=[neuron0,neuron1,neuron2,neuron3,neuron4,neuron5,neuron6,neuron7]
manual_encoding=[manual_encoding for i in range(32)]
manual_encoding=np.array(manual_encoding)
temp=decoder.predict(np.expand_dims(manual_encoding,axis=0))
smiles_generated=OneHotDecoding_helper(temp[0],key_value)
print(smiles_generated)
if len(smilite.get_zincid_from_smile(smiles_generated))>0:
  print(smilite.get_zincid_from_smile(smiles_generated))
```

```
OOCnnccccccccccc1cccc1c1N1CccNN
```

8.6 Neuron 6 adds brackets and 7 adds more combinations.

Molecules generated by Manual tweaking of latent space

1. CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
2. OOOCCCCC111111ccccccc1111=(1)c11
3. OCCCOOOCc1cccccc1111c11ccccN1o4o
4. CCCCCCccccccNO1N1[CC[[CH[ooC5C52
5. OOCnnccccccccccc1cccc1c1N1CccNN

The above molecules were not flagged by Smilite as invalid.

8.7 Molecules generated with Valid ZINC 15 ID

[274]:
```
new_molecules
```

[274]:

	Generated Molecules	Zinc_ID
0	CCCCCCCCCCCCCCCCCCC(=O)c1ccccc1	[ZINC000115464572]
1	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC	[ZINC000006920423]
2	CCCCCCCCCCCCCCCCCCCCCCCCCCC(CO)O	[ZINC000103820528, ZINC000103820533]
3	c1ccc(CCCCCCCCCCCCCCc2ccccc2)cc1	[ZINC000140966601]

9 Latent space of above variables

[278]:
```
# Of CCCCCCCCCCCCCCCCCCC(=O)c1ccccc1
print(encoder_values[0][0])
```

```
[0.        0.        3.117608  4.28397   3.346471  2.7407904 9.601776
 2.3084958]
```

[279]:
```
#Of CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
print(encoder_values[1][0])
```

```
[ 0.         0.         2.7760482  3.5332496  2.9913094  1.4493
 11.649847   2.6623216]
```

[280]:
```
#Of CCCCCCCCCCCCCCCCCCCCCCCCCCCC(CO)O
print(encoder_values[2][0])
```

```
[ 0.         0.         2.4487445  3.6849833  2.9494824  1.5021274
 11.464967   2.4970162]
```

[281]:
```
# Of c1ccc(CCCCCCCCCCCCCCc2ccccc2)cc1
print(encoder_values[3][0])
```

```
[ 0.         0.         8.36374    2.0634527 17.87391    2.0587478
 20.242287  24.374287 ]
```

[ ]:
```

```