

#@title Drug Design

## Drug Design

```
# from google.colab import drive
# drive.mount("/content/drive")

# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras import layers
from keras.models import Model
from keras import metrics
from keras import backend as K
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
import warnings
import matplotlib.pyplot as plt
from tqdm import tqdm
import random

# Set environment variables and clear sessions
K.clear_session()
np.random.seed(237)
warnings.filterwarnings("ignore")
tf.compat.v1.disable_eager_execution()
%matplotlib inline

## Import Dataset from drive for Smile representation of drug molecules:
## Source for data: https://zinc.docking.org/substances/subsets/AA/
data = pd.read_csv('/content/drive/MyDrive/Dataset/MoleculeSythesis/dataset/AA.csv')
data.head()
```

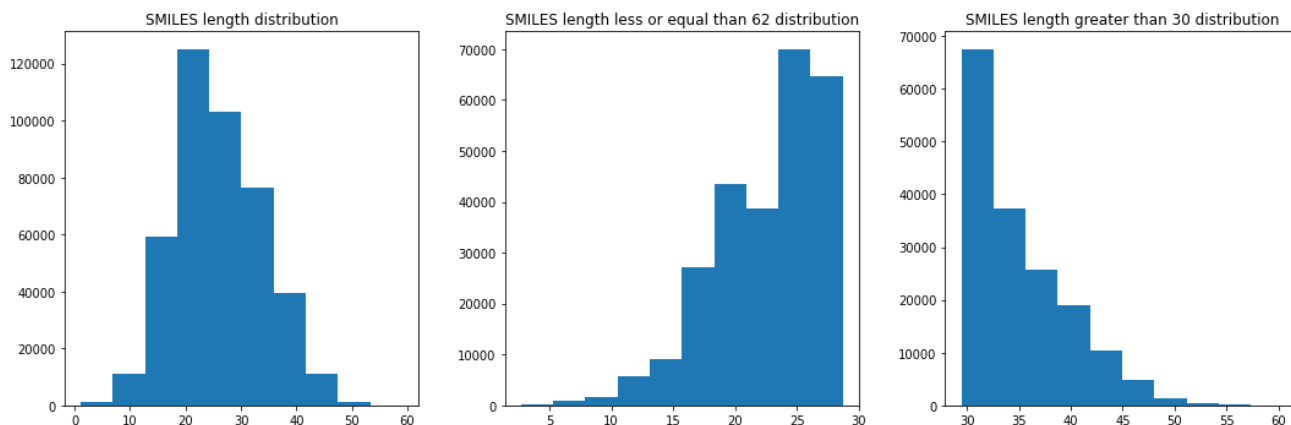
	zinc_id	smiles
0	ZINC000000008151	<chem>C[C@H]1[C@@H](O)[C@H](CO)O[C@@H](O)[C@@H]1N</chem>
1	ZINC000000008153	<chem>CC[C@@H]1[C@@H](N)[C@@H](O)O[C@@H](CO)[C@@H]1O</chem>
2	ZINC000000008155	<chem>CC1(C)[C@@H](N)[C@@H](O)O[C@@H](CO)[C@@H]1O</chem>
3	ZINC000000018276	<chem>CS[C@@H]1CN[C@@H](CO)[C@H](O)[C@H]1O</chem>
4	ZINC000000018279	<chem>CS[C@@H]1[C@@H](O)CN[C@@H](CO)[C@@H]1O</chem>

```
#### Plot distribution of smile molecules vs the molecule length
fig, axes = plt.subplots(nrows = 1, ncols = 3, figsize = (15, 5))
axes[0].set_title("SMILES length distribution")
axes[0].hist(data['smiles'].str.len(), align = 'left')
axes[1].set_title("SMILES length less or equal than 62 distribution")
axes[1].hist(data[data['smiles'].str.len() <= 62]['smiles'].str.len(), align = 'left')
```

```

axes[2].set_title("SMILES length greater than 30 distribution")
axes[2].hist(data[data['smiles'].str.len() > 30]['smiles'].str.len(), align = 'left')
fig.tight_layout()
plt.show()

```



```
# Super set of characters used for smile generations
```

```

SMILES_CHARS = [ ' ',
                  '#', '%', '(', ')', '+', '-', '.', '/',
                  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                  '=', '@',
                  'A', 'B', 'C', 'F', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P',
                  'R', 'S', 'T', 'V', 'X', 'Z',
                  '[', '\\', ']',
                  'a', 'b', 'c', 'e', 'g', 'i', 'l', 'n', 'o', 'p', 'r', 's',
                  't', 'u'
                ]

```

```
#### Create dictionaries from character set for encoding and decoding smiles
```

```

encoder_dict = dict( (c,i) for i,c in enumerate( SMILES_CHARS ) )
decoder_dict = dict( (i,c) for i,c in enumerate( SMILES_CHARS ) )
print(encoder_dict)
print(decoder_dict)

```

```

{' ': 0, '#': 1, '%': 2, '(': 3, ')': 4, '+': 5, '-': 6, '.': 7, '/': 8, '0': 9,
'1': 10, '2': 11, '3': 12, '4': 13, '5': 14, '6': 15, '7': 16, '8': 17, '9': 18,
'=': 19, '@': 20, 'A': 21, 'B': 22, 'C': 23, 'F': 24, 'H': 25, 'I': 26, 'K': 27, 'L': 28, 'M': 29, 'N': 30, 'O': 31, 'P': 32,
'R': 33, 'S': 34, 'T': 35, 'V': 36, 'X': 37, 'Z': 38, '[': 39, '\\': 40, ']': 41, 'a': 42, 'b': 43, 'c': 44, 'e': 45, 'g': 46, 'i': 47, 'l': 48, 'n': 49, 'o': 50, 'p': 51, 'r': 52, 's': 53,
't': 54, 'u': 55}

```

```

smiles_data = data['smiles'][:250000]
smiles_data = np.array(smiles_data).reshape(-1)
print('Number of mols: '+str(len(smiles_data)))
idx = [i for i, x in enumerate(smiles_data) if len(x)<=120]
print('Number of valid mols: '+str(len(idx)))
smiles_data = smiles_data[idx]
print('Getting a unique character set...')

```

```

Number of mols: 250000
Number of valid mols: 250000
Getting a unique character set...

```

```

char_set = set()
for i in tqdm(range(len(smiles_data))):
    smiles_data[i] = smiles_data[i].ljust(62)
    char_set = char_set.union(set(smiles_data[i]))
char_set_list = sorted(list(char_set))
print('Number of characters: '+str(len(char_set_list)))

```

```

100%|██████████| 250000/250000 [00:00<00:00, 396205.17it/s]Number of character

```

```

def one_hot_encoder( smiles ):
    X = np.zeros( ( 62, len( SMILES_CHARS ) ) )
    for i, c in enumerate( smiles ):
        X[i, encoder_dict[c]] = 1
    return X

```

```

def one_hot_decoder( X ):
    smi = ''
    X = X.argmax( axis=-1 )
    for i in X:
        smi += decoder_dict[i]
    return smi

```

```

x = []
for i in data['smiles'][:500]:
    x.append(one_hot_encoder(i))
arr = np.array(x)

```

```

arr = arr.reshape(-1,62,56,1)
arr.shape

```

```

(500, 62, 56, 1)

```

```

#### Basic VAE Keras implementatin used as reference for encoder and decoder layer
#### Link: https://zinc.docking.org/substances/subsets/AA/

```

```

img_shape = (62, 56,1)
latent_dim = 2

```

```

input_img = keras.Input(shape=img_shape)

```

```

x = layers.Conv1D(32, 7,
                  padding='same',
                  activation='relu',
                  )(input_img)

```

```

x = layers.Conv2D(128, 3,
                  padding='same',

```

```

        padding='same',
        activation='relu',
        strides=(2, 2))(x)

x = layers.Conv2D(128, 3,
                  padding='same',
                  activation='relu')(x)

x = layers.Conv2D(128, 3,
                  padding='same',
                  activation='relu')(x)

shape_before_flattening = K.int_shape(x)

x = layers.Flatten()(x)
x = layers.Dense(32, activation='relu')(x)

z_mu = layers.Dense(latent_dim)(x)
z_log_sigma = layers.Dense(latent_dim)(x)

def sampling(args):
    z_mu, z_log_sigma = args
    epsilon = K.random_normal(shape=(K.shape(z_mu)[0], latent_dim),
                               mean=0., stddev=1.)
    return z_mu + K.exp(z_log_sigma) * epsilon

z = layers.Lambda(sampling)([z_mu, z_log_sigma])

decoder_input = layers.Input(K.int_shape(z)[1:])

x = layers.Dense(np.prod(shape_before_flattening[1:]),
                  activation='relu')(decoder_input)

x = layers.Reshape(shape_before_flattening[1:])(x)

x = layers.Conv2DTranspose(32, 3,
                           padding='same',
                           activation='relu',
                           strides=(2, 2))(x)

x = layers.Conv2D(1, 3,
                  padding='same',
                  activation='sigmoid')(x)

decoder = Model(decoder_input, x)

z_decoded = decoder(z)

class CustomVariationalLayer(keras.layers.Layer):
    def vae_loss(self, x, z_decoded):
        x = K.flatten(x)
        z_decoded = K.flatten(z_decoded)
        xent_loss = keras.metrics.binary_crossentropy(x, z_decoded) #Recin loss
        kl_loss = -5e-4 * K.mean(1 + z_log_sigma - K.square(z_mu) - K.exp(z_log_sigma))
        return xent_loss + kl_loss

```

```
def call(self, inputs):
    x = inputs[0]
    z_decoded = inputs[1]
    loss = self.vae_loss(x, z_decoded)
    self.add_loss(loss, inputs=inputs)
    return x
```

```
def call(self, inputs):
    x = inputs[0]
    z_decoded = inputs[1]
    loss = self.vae_loss(x, z_decoded)
    self.add_loss(loss, inputs=inputs)
    return x

y = CustomVariationalLayer()([input_img, z_decoded])
```

```
# VAE model statement
vae = Model(input_img, y)

vae.compile(optimizer='rmsprop', loss=None)

vae.summary()
```

WARNING:tensorflow:Output custom\_variational\_layer missing from loss dictionary  
Model: "model\_1"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 62, 56, 1)]	0	
conv1d (Conv1D)	(None, 62, 56, 32)	256	input_1[0][0]
conv2d (Conv2D)	(None, 31, 28, 128)	36992	conv1d[0][0]
conv2d_1 (Conv2D)	(None, 31, 28, 128)	147584	conv2d[0][0]
conv2d_2 (Conv2D)	(None, 31, 28, 128)	147584	conv2d_1[0][0]
flatten (Flatten)	(None, 111104)	0	conv2d_2[0][0]
dense (Dense)	(None, 32)	3555360	flatten[0][0]
dense_1 (Dense)	(None, 2)	66	dense[0][0]
dense_2 (Dense)	(None, 2)	66	dense[0][0]
lambda (Lambda)	(None, 2)	0	dense_1[0][0] dense_2[0][0]
model (Functional)	(None, 62, 56, 1)	370497	lambda[0][0]
custom_variational_layer (Custo	(None, 62, 56, 1)	0	input_1[0][0] model[0][0]
Total params: 4,258,405			
Trainable params: 4,258,405			
Non-trainable params: 0			

```
vae.fit(x=arr, y=None,
        shuffle=False,
        epochs=120,
```

batch\_size=16

)

Train on 500 samples

Epoch 1/120

500/500 [=====] - 1s 2ms/sample - loss: 1568276.3942

Epoch 2/120

500/500 [=====] - 0s 613us/sample - loss: 0.0442

Epoch 3/120

500/500 [=====] - 0s 611us/sample - loss: 0.0378

Epoch 4/120

500/500 [=====] - 0s 605us/sample - loss: 0.0358

Epoch 5/120

500/500 [=====] - 0s 618us/sample - loss: 0.0348

Epoch 6/120

500/500 [=====] - 0s 618us/sample - loss: 0.0342

Epoch 7/120

500/500 [=====] - 0s 615us/sample - loss: 0.0338

Epoch 8/120

500/500 [=====] - 0s 621us/sample - loss: 0.0335

Epoch 9/120

500/500 [=====] - 0s 618us/sample - loss: 0.0333

Epoch 10/120

500/500 [=====] - 0s 627us/sample - loss: 0.0330

Epoch 11/120

500/500 [=====] - 0s 637us/sample - loss: 0.0326

Epoch 12/120

500/500 [=====] - 0s 618us/sample - loss: 0.0325

Epoch 13/120

500/500 [=====] - 0s 625us/sample - loss: 0.0322

Epoch 14/120

500/500 [=====] - 0s 639us/sample - loss: 0.0324

Epoch 15/120

500/500 [=====] - 0s 625us/sample - loss: 0.0320

Epoch 16/120

500/500 [=====] - 0s 615us/sample - loss: 0.0322

Epoch 17/120

500/500 [=====] - 0s 611us/sample - loss: 0.0319

Epoch 18/120

500/500 [=====] - 0s 617us/sample - loss: 0.0316

Epoch 19/120

500/500 [=====] - 0s 612us/sample - loss: 0.0314

Epoch 20/120

500/500 [=====] - 0s 609us/sample - loss: 0.0313

Epoch 21/120

500/500 [=====] - 0s 611us/sample - loss: 0.0312

Epoch 22/120

500/500 [=====] - 0s 610us/sample - loss: 0.0310

Epoch 23/120

500/500 [=====] - 0s 608us/sample - loss: 0.0309

Epoch 24/120

500/500 [=====] - 0s 633us/sample - loss: 0.0307

Epoch 25/120

500/500 [=====] - 0s 621us/sample - loss: 0.0306

Epoch 26/120

500/500 [=====] - 0s 615us/sample - loss: 0.0305

Epoch 27/120

500/500 [=====] - 0s 618us/sample - loss: 0.0303

Epoch 28/120

500/500 [=====] - 0s 614us/sample - loss: 0.0304

Epoch 29/120

500/500 [=====] - 0s 613us/sample - loss: 0.0299

```
### Predicting Molecules from Latent space
```

```
result = []
```

```
for i in random.sample(range(1, 10000000), 100):  
    for j in random.sample(range(1, 10000000), 100):  
        sample_vector = np.array([[i,j]])  
        res = decoder.predict(sample_vector)  
        res = res[0].reshape(62,56)  
        result.append(one_hot_decoder(res).replace(' ',''))
```

```
set(result)
```

