Start coding or generate with AI.

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Flatten, TimeDistributed, Input
from keras.optimizers import Adam, SGD
import os


# Ensure output directory exists
if not os.path.exists(os.getcwd() + '/output/'):
    os.makedirs(os.getcwd() + '/output/')


# Shuffling function for 3D arrays
def shuffle3D(arr):
    for a in arr:
        np.random.shuffle(a)


# Add time steps to features
def dimX(x, ts):
    x = np.asarray(x)
    newX = []
    for i, c in enumerate(x):
        newX.append([c] * ts)
    return np.array(newX)


# Add time steps to target
def dimY(Y, ts, chars, char_idx):
    temp = np.zeros((len(Y), ts, len(chars)), dtype=np.bool_)
    for i, c in enumerate(Y):
        for j, s in enumerate(c):
            temp[i, j, char_idx[s]] = 1
    return np.array(temp)


# Sequence prediction with argmax
def prediction(preds):
```

```
    y_pred = []
    for i, c in enumerate(preds):
        y_pred.append([np.argmax(j) for j in c])
    return np.array(y_pred)


# Sequence to text conversion
def seq_txt(y_pred, idx_char):
    newY = []
    for i, c in enumerate(y_pred):
        newY.append([idx_char[j] for j in c])
    return np.array(newY)


# Convert SMILES output from the model
def smiles_output(s):
    smiles = []
    for i in s:
        smiles.append(''.join(str(k) for k in i))
    return smiles


# Generator model
def Gen(x_dash, y_dash):
    G = Sequential()
    # Define input shape using Input layer to avoid the warning
    G.add(Input(shape=(x_dash.shape[1], x_dash.shape[2])))  # Input layer
    G.add(TimeDistributed(Dense(x_dash.shape[2])))
    G.add(LSTM(216, return_sequences=True))
    G.add(Dropout(0.3))
    G.add(LSTM(216, return_sequences=True))
    G.add(Dropout(0.3))
    G.add(LSTM(216, return_sequences=True))
    G.add(TimeDistributed(Dense(y_dash.shape[2], activation='softmax')))
    G.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=2e-4))
    return G


# Discriminator model
def Dis(y_dash):
    D = Sequential()
    # Define input shape using Input layer to avoid the warning
```

```python
    D.add(Input(shape=(y_dash.shape[1], y_dash.shape[2])))  # Input layer
    D.add(TimeDistributed(Dense(y_dash.shape[2])))
    D.add(LSTM(216, return_sequences=True))
    D.add(Dropout(0.3))
    D.add(LSTM(60, return_sequences=True))
    D.add(Flatten())
    D.add(Dense(1, activation='sigmoid'))
    D.compile(loss='binary_crossentropy', optimizer=SGD(learning_rate=0.001))
    return D


# GAN model
def Gan(G, D):
    GAN = Sequential()
    GAN.add(G)  # Add Generator
    D.trainable = False
    GAN.add(D)  # Add Discriminator
    GAN.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=2e-4))
    return GAN


# Train Discriminator
def trainDis(D, data, x_dash, y_dash, mc=None):
    if mc is None:
        fake_data = G.predict(x_dash)
        targets = np.zeros(x_dash.shape[0]).astype(int)
        Dloss = D.fit(fake_data, targets, epochs=1, batch_size=32)
    else:
        fake_ydata = np.copy(y_dash)
        shuffle3D(fake_ydata)
        targets = np.zeros(x_dash.shape[0]).astype(int)
        Dloss = D.fit(fake_ydata, targets, epochs=1, batch_size=32)

    return Dloss.history['loss'][0]


# Train GAN
def trainGAN(GAN, x_dash):
    target = np.ones(x_dash.shape[0]).astype(int)
    gan_loss = GAN.fit(x_dash, target, epochs=1, batch_size=32)
    return gan_loss.history['loss'][0]
```

```
from google.colab import files
files.upload();
```

Choose Files Smiles_data.csv

- **Smiles_data.csv**(text/csv) - 949445 bytes, last modified: 7/12/2024 - 100% done

```
##read csv file
data = pd.read_csv("Smiles_data.csv")
data = data.sample(frac=1).reset_index(drop=True)
```

```
Y=data.SMILES
Y.head()
```

	SMILES
0	O=c1ccc(Br)cn1C/C([O-])=N/S(=O)(=O)CCCF
1	C#CCOc1ccc(NC(=O)C(=O)NC[C@@H]2CCC[C@@H](O)C2)cc1
2	Cc1cccc(NC(=S)NNC(=O)c2cnc(-c3cnn(C)c3)s2)c1C
3	O=C(Cc1ccc(Br)cc1)Nc1nnc(C2CCCCC2)s1
4	O=C1O[C@H](c2ccccc2)Cc2cc(C(=O)N3CCCCC3)ccc21

```
# Check for NaN values and handle them (you can choose to drop or fill them)
data = data.dropna(subset=['SMILES'])  # Drop rows where 'SMILES' is NaN

# Ensure that X columns are numeric and handle errors during conversion
X = data.iloc[:, 1:7]

# Check if any values are non-numeric or NaN
# Optionally fill NaNs with a specific value like 0 or the column mean
X = X.apply(pd.to_numeric, errors='coerce').fillna(0).astype(int)

# Now you can safely work with X
```

```
print(X.head())
```

```
   Unnamed: 1  Unnamed: 2  Unnamed: 3  Unnamed: 4  Unnamed: 5  Unnamed: 6
0           0           0           0           0           0           0
1           0           0           0           0           0           0
2           0           0           0           0           0           0
3           0           0           0           0           0           0
4           0           0           0           0           0           0
```

```
# Padding SMILES to equal length
maxY = Y.str.len().max()
y = Y.str.ljust(maxY, fillchar='|')


# CharToIndex and IndexToChar functions
chars = sorted(set("".join(y.values.flatten())))
char_idx = {c: i for i, c in enumerate(chars)}
idx_char = {i: c for i, c in enumerate(chars)}

ts = y.str.len().max()
y_dash = dimY(y, ts, chars, char_idx)
x_dash = dimX(X, ts)


# Initialize models
G = Gen(x_dash, y_dash)
D = Dis(y_dash)
GAN = Gan(G, D)


# Pre-training Discriminator
for i in range(20):
    shuffleData = np.random.permutation(y_dash)
    disloss = trainDis(D, shuffleData, x_dash, y_dash)
    print(f"Pre Training Discriminator Loss: {disloss}")
```

```
Pre Training Discriminator Loss: [illegible]
469/469 ━━━━━━━━━━━━━━━━━━━━ 141s 301ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 60s 128ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6897923946380615
469/469 ━━━━━━━━━━━━━━━━━━━━ 142s 303ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 60s 128ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.689787745475769
469/469 ━━━━━━━━━━━━━━━━━━━━ 138s 295ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 126ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.689771831035614
469/469 ━━━━━━━━━━━━━━━━━━━━ 136s 290ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 126ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6897746324539185
469/469 ━━━━━━━━━━━━━━━━━━━━ 135s 288ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 61s 129ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6897813677787781
469/469 ━━━━━━━━━━━━━━━━━━━━ 141s 302ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6897804141044617
469/469 ━━━━━━━━━━━━━━━━━━━━ 139s 296ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6898065805435181
469/469 ━━━━━━━━━━━━━━━━━━━━ 138s 295ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 58s 123ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.689788281917572
469/469 ━━━━━━━━━━━━━━━━━━━━ 138s 293ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 58s 123ms/step - loss: 0.6897
Pre Training Discriminator Loss: 0.6897650957107544
469/469 ━━━━━━━━━━━━━━━━━━━━ 139s 297ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 60s 128ms/step - loss: 0.6897
Pre Training Discriminator Loss: 0.6897476315498352
469/469 ━━━━━━━━━━━━━━━━━━━━ 139s 297ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 58s 125ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6897788047790527
469/469 ━━━━━━━━━━━━━━━━━━━━ 139s 297ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 58s 124ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6897987127304077
469/469 ━━━━━━━━━━━━━━━━━━━━ 136s 290ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.6897578835487366
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 138s 294ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 62s 132ms/step - loss: 0.6898
Pre Training Discriminator Loss: 0.689799427986145
469/469 ━━━━━━━━━━━━━━━━━━━━ 137s 291ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 60s 129ms/step - loss: 0.6898
```

```
# Train GAN
episodes = 10
for episode in range(episodes):
    print(f"Epoch {episode}/{episodes}")
    disloss = trainDis(D, y_dash, x_dash, y_dash)
    disloss_mc = trainDis(D, y_dash, x_dash, y_dash, mc="mc")
    ganloss = trainGAN(GAN, x_dash)

    print(f"D loss={disloss} | D (mc) loss={disloss_mc} | GAN loss={ganloss}")
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 139s 295ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 62s 131ms/step - loss: 0.6898
469/469 ━━━━━━━━━━━━━━━━━━━━ 67s 139ms/step - loss: 0.6624
469/469 ━━━━━━━━━━━━━━━━━━━━ 698s 1s/step - loss: 0.0000e+00
D loss=0.689791202545166 | D (mc) loss=0.6623251438140869 | GAN loss=0.0
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 138s 294ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 62s 133ms/step - loss: 0.7854
469/469 ━━━━━━━━━━━━━━━━━━━━ 61s 130ms/step - loss: 0.6624
469/469 ━━━━━━━━━━━━━━━━━━━━ 683s 1s/step - loss: 0.0000e+00
D loss=0.7853882312774658 | D (mc) loss=0.6624612212181091 | GAN loss=0.0
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 139s 296ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 135ms/step - loss: 0.7867
469/469 ━━━━━━━━━━━━━━━━━━━━ 64s 136ms/step - loss: 0.6624
469/469 ━━━━━━━━━━━━━━━━━━━━ 684s 1s/step - loss: 0.0000e+00
D loss=0.7867259383201599 | D (mc) loss=0.6624104976654053 | GAN loss=0.0
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 136s 289ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 61s 131ms/step - loss: 0.7872
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 135ms/step - loss: 0.6623
469/469 ━━━━━━━━━━━━━━━━━━━━ 678s 1s/step - loss: 0.0000e+00
D loss=0.7871255278587341 | D (mc) loss=0.6623762845993042 | GAN loss=0.0
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 137s 292ms/step
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 62s 133ms/step - loss: 0.6623
469/469 ━━━━━━━━━━━━━━━━━━━━ 674s 1s/step - loss: 0.0000e+00
D loss=0.7874006032943726 | D (mc) loss=0.6623557209968567 | GAN loss=0.0
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 138s 294ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 135ms/step - loss: 0.7875
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 133ms/step - loss: 0.6626
469/469 ━━━━━━━━━━━━━━━━━━━━ 686s 1s/step - loss: 0.0000e+00
D loss=0.7874447703361511 | D (mc) loss=0.6624957323074341 | GAN loss=0.0
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 136s 291ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 135ms/step - loss: 0.7877
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 134ms/step - loss: 0.6623
469/469 ━━━━━━━━━━━━━━━━━━━━ 672s 1s/step - loss: 0.0000e+00
D loss=0.7876733541488647 | D (mc) loss=0.662282407283783 | GAN loss=0.0
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 137s 291ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 134ms/step - loss: 0.7878
469/469 ━━━━━━━━━━━━━━━━━━━━ 61s 130ms/step - loss: 0.6625
469/469 ━━━━━━━━━━━━━━━━━━━━ 669s 1s/step - loss: 0.0000e+00
D loss=0.7877541184425354 | D (mc) loss=0.6624325513839722 | GAN loss=0.0
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 137s 292ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 135ms/step - loss: 0.7879
469/469 ━━━━━━━━━━━━━━━━━━━━ 63s 134ms/step - loss: 0.6625
469/469 ━━━━━━━━━━━━━━━━━━━━ 668s 1s/step - loss: 0.0000e+00
D loss=0.7877466082572937 | D (mc) loss=0.662394642829895 | GAN loss=0.0
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 137s 291ms/step
469/469 ━━━━━━━━━━━━━━━━━━━━ 61s 130ms/step - loss: 0.7879
469/469 ━━━━━━━━━━━━━━━━━━━━ 61s 130ms/step - loss: 0.6625
385/469 ━━━━━━━━━━━━━━━━━━━━ 2:00 1s/step - loss: 0.0000e+00
```

```
# Optionally save model weights
   if episode % 10 == 0:
        G.save(os.path.join(os.getcwd(), 'output', 'Gen.h5'))
        D.save(os.path.join(os.getcwd(), 'output', 'Dis.h5'))
        GAN.save(os.path.join(os.getcwd(), 'output', 'Gan.h5'))
```

```
    # Sample predictions
    if episode % 100 == 0:
        print("Predicting Molecule")
        x_pred = [[0, 0, 0, 1, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1]]
        x_pred = dimX(x_pred, ts)
```

```
    preds = G.predict(x_pred)
    y_pred = prediction(preds)
    y_pred = seq_txt(y_pred, idx_char)
    s = smiles_output(y_pred)
    print(s)
```