# GAN network to generate new image:

A generative adversarial network (GAN) is a class of machine learning systems invented by Ian Goodfellow and his colleagues in 2014.Two neural networks contest with each other in a game (in the sense of game theory, often but not always in the form of a zero-sum game). Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

### Dataset: MNIST

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. Image size are 28x 28.

### Source Code:

# -\*- coding: utf-8 -\*-

"""DCGAN.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1rtJ1Gp22aUlWTLnLBoOlOd-mWpl49w\_G

"""

import tensorflow as tf

from keras.datasets import mnist

import time

import matplotlib.pyplot as plt

from IPython import display

import os

from IPython.display import Image

import numpy as np

(xtrain, ytrain) , (xtest,ytest) = mnist.load\_data()

print(xtrain.shape)

print(xtest.shape)

xtrain = xtrain.reshape(xtrain.shape[0], 28, 28, 1).astype('float32')

xtrain = (xtrain - 127.5) / 127.5

print(xtrain.shape)

batchSize = 256

parameterInsideAdam = 1e-4

trainSet = tf.data.Dataset.from\_tensor\_slices(xtrain)

trainSet = trainSet.shuffle(xtrain.shape[0])

trainSet = trainSet.batch(batchSize)

def ganGeneartor():

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(7\*7\*256, use\_bias=False, input\_shape=(100,)))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Reshape((7, 7, 256)))

assert model.output\_shape == (None, 7, 7, 256) # Note: None is the batch size

model.add(tf.keras.layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use\_bias=False))

assert model.output\_shape == (None, 7, 7, 128)

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use\_bias=False))

assert model.output\_shape == (None, 14, 14, 64)

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use\_bias=False, activation='tanh'))

assert model.output\_shape == (None, 28, 28, 1)

return model

generator = ganGeneartor()

def ganDiscriminator():

model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',

input\_shape=[28, 28, 1]))

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Dropout(0.3))

model.add(tf.keras.layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Dropout(0.3))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(1))

return model

discriminator = ganDiscriminator()

def geneartorLoss(generatedOutput):

# print("generator type: ",type(generatedOutput))

generatorLoss = tf.keras.losses.BinaryCrossentropy(from\_logits=True)(tf.ones\_like(generatedOutput), generatedOutput)

return generatorLoss

def discriminatorLoss(generatedOutput, realOutput):

# print("generator type: ",type(generatedOutput))

# print("discriminator type: ",type(generatedOutput))

generatedOutputLoss = tf.keras.losses.BinaryCrossentropy(from\_logits=True)(tf.zeros\_like(generatedOutput), generatedOutput)

realOutputLoss = tf.keras.losses.BinaryCrossentropy(from\_logits=True)(tf.ones\_like(realOutput), realOutput)

return (generatedOutputLoss + realOutputLoss)

noise\_dim = 100

num\_examples\_to\_generate = 16

seed = tf.random.normal([num\_examples\_to\_generate, noise\_dim])

generator = ganGeneartor()

discriminator = ganDiscriminator()

generator\_optimizer = tf.keras.optimizers.Adam(1e-4)

discriminator\_optimizer = tf.keras.optimizers.Adam(1e-4)

# This annotation causes the function to be "compiled".

@tf.function

def trainInBatch(imageBatch):

noise = tf.random.normal([batchSize, noise\_dim])

with tf.GradientTape() as generatorTape, tf.GradientTape() as discriminatorTape:

geneartedImage = generator(noise,training = True)

realOutput = discriminator(imageBatch,training =True)

# print("realOutput:",realOutput)

fakingOutput = discriminator(geneartedImage, training = True)

# print("fakeOutput:",realOutput)

generator\_loss = geneartorLoss(fakingOutput)

discriminator\_loss = discriminatorLoss(fakingOutput, realOutput)

generator\_optimizer.apply\_gradients(zip(generatorTape.gradient(generator\_loss,generator.trainable\_variables) , generator.trainable\_variables )) #For Backpropagation

discriminator\_optimizer.apply\_gradients(zip(discriminatorTape.gradient(discriminator\_loss,discriminator.trainable\_variables) , discriminator.trainable\_variables )) #For Backpropagation

def generateAndSave(model, epoch, test\_input):

predictions = model(test\_input,training = False)

# generatedImage = np.asarray(predictions[0]\*127.5 + 127.5)

# print(generatedImage.shape)

fig = plt.figure(figsize=(4,4))

for i in range(predictions.shape[0]):

plt.subplot(4, 4, i+1)

generatedImage = tf.keras.preprocessing.image.array\_to\_img(predictions[i]\*127.5 + 127.5, data\_format=None, scale=True, dtype='float32')

plt.imshow(np.asarray(generatedImage), cmap='gray')

plt.axis('off')

plt.savefig("image\_at\_epoch\_"+ str(epoch)+".png")

plt.show()

def train(data, epochs):

# originalImage = tf.keras.preprocessing.image.array\_to\_img(xtrain[0], data\_format=None, scale=True, dtype='float32')

# plt.imshow(np.asarray(originalImage),cmap="gray")

# plt.show()

# originalImage.save("originalImage.png","PNG")

# originalImageLocationString = tf.io.read\_file("originalImage.png")

# originalImage = tf.io.decode\_png(originalImageLocationString)

# # print(type(originalImage))

for epoch in range(epochs):

startTime = time.time()

for imageBatch in data:

trainInBatch(imageBatch)

# seed = tf.random.normal([num\_examples\_to\_generate, noise\_dim])

# predictions = generator(seed,training = False)

# generatedImage = tf.keras.preprocessing.image.array\_to\_img(predictions[0]\*127.5 + 127.5, data\_format=None, scale=True, dtype='float32')

# plt.imshow(np.asarray(generatedImage),cmap="gray")

# plt.show()

# generatedImageLocation = "Epoch-"+str(epoch+1)+".png"

# generatedImage.save(generatedImageLocation)

# generatedImageLocationString = tf.io.read\_file(generatedImageLocation)

# generatedImage = tf.io.decode\_png(generatedImageLocationString)

# # print(type(generatedImage))

# ssim = tf.image.ssim(originalImage, generatedImage, max\_val=1.0, filter\_size=4,

# filter\_sigma=1.5, k1=0.01, k2=0.03)

# print("Epoch : ", epoch+1,",", " training time : ",time.time()-startTime, ",", "SSIM : ",ssim)

# plt.imshow(xtrain[0].reshape(28,28), cmap="gray")

# os.remove(generatedImageLocation)

# generated\_image = generator(seed, training = False)

# print("Discriminator Loss: ",discriminator(generated\_image))

# loss = discriminator(generated\_image)

print("Epoch : ", epoch+1,",", " training time : ",time.time()-startTime)

# loss = "discriminator loss for 16 example are: " + str(tf.print(loss[0]))

# print(loss)

print("---------------------Generated Image-----------------------------------------")

generateAndSave(generator,epochs,seed)

print("---------------------Original Image-----------------------------------------")

fig = plt.figure(figsize=(4,4))

for i in range(16):

plt.subplot(4, 4, i+1)

originalImage = tf.keras.preprocessing.image.array\_to\_img(xtrain[i]\*127.5 + 127.5, data\_format=None, scale=True, dtype='float32')

plt.imshow(np.asarray(originalImage), cmap='gray')

plt.axis('off')

plt.savefig("originalImage.png")

plt.show()

# generated\_image = generator(seed, training = False)

# print("Final Discriminator Loss for 16 Generated Image: ",discriminator(generated\_image))

epochs = 100

train(trainSet,epochs)

from PIL import Image

generatedImage = Image.open("generated.PNG")

generatedImage = generatedImage.resize((28,28))

originalImage = Image.open("original.PNG")

originalImage = originalImage.resize((28,28))

from skimage.metrics import structural\_similarity as ssim

ssimValue , diff = ssim(np.asarray(originalImage), np.asarray(generatedImage),multichannel=True, full =True)

print("SSIMvalue: ",ssimValue)

### Results:

Epoch : 1 , training time : 12.238376140594482

Epoch : 2 , training time : 10.400644063949585

Epoch : 3 , training time : 10.52396821975708

Epoch : 4 , training time : 10.662012100219727

Epoch : 5 , training time : 10.79157042503357

Epoch : 6 , training time : 10.706889867782593

Epoch : 7 , training time : 10.615949630737305

Epoch : 8 , training time : 10.607760667800903

Epoch : 9 , training time : 10.539034843444824

Epoch : 10 , training time : 10.592292070388794

Epoch : 11 , training time : 10.573237419128418

Epoch : 12 , training time : 10.641231536865234

Epoch : 13 , training time : 10.611907005310059

Epoch : 14 , training time : 10.605278015136719

Epoch : 15 , training time : 10.592592716217041

Epoch : 16 , training time : 10.622884273529053

Epoch : 17 , training time : 10.575559616088867

Epoch : 18 , training time : 10.580503463745117

Epoch : 19 , training time : 10.56233549118042

Epoch : 20 , training time : 10.577748775482178

Epoch : 21 , training time : 10.587005853652954

Epoch : 22 , training time : 10.571271419525146

Epoch : 23 , training time : 10.614445447921753

Epoch : 24 , training time : 10.577542066574097

Epoch : 25 , training time : 10.586507320404053

Epoch : 26 , training time : 10.632090330123901

Epoch : 27 , training time : 10.620543718338013

Epoch : 28 , training time : 10.579862594604492

Epoch : 29 , training time : 10.593589067459106

Epoch : 30 , training time : 10.595964670181274

Epoch : 31 , training time : 10.613166332244873

Epoch : 32 , training time : 10.59261441230774

Epoch : 33 , training time : 10.603117942810059

Epoch : 34 , training time : 10.590378284454346

Epoch : 35 , training time : 10.617982864379883

Epoch : 36 , training time : 10.649446964263916

Epoch : 37 , training time : 10.636476039886475

Epoch : 38 , training time : 10.643375158309937

Epoch : 39 , training time : 10.606611251831055

Epoch : 40 , training time : 10.59779667854309

Epoch : 41 , training time : 10.613938093185425

Epoch : 42 , training time : 10.605061054229736

Epoch : 43 , training time : 10.608515977859497

Epoch : 44 , training time : 10.607630252838135

Epoch : 45 , training time : 10.60928988456726

Epoch : 46 , training time : 10.606380701065063

Epoch : 47 , training time : 10.606993436813354

Epoch : 48 , training time : 10.615205764770508

Epoch : 49 , training time : 10.60983395576477

Epoch : 50 , training time : 10.594130277633667

Epoch : 51 , training time : 10.61570405960083

Epoch : 52 , training time : 10.615573406219482

Epoch : 53 , training time : 10.609174728393555

Epoch : 54 , training time : 10.595293283462524

Epoch : 55 , training time : 10.619438409805298

Epoch : 56 , training time : 10.605151653289795

Epoch : 57 , training time : 10.60616397857666

Epoch : 58 , training time : 10.611514806747437

Epoch : 59 , training time : 10.641344785690308

Epoch : 60 , training time : 10.609029054641724

Epoch : 61 , training time : 10.615436553955078

Epoch : 62 , training time : 10.592969179153442

Epoch : 63 , training time : 10.600862979888916

Epoch : 64 , training time : 10.59118366241455

Epoch : 65 , training time : 10.59646463394165

Epoch : 66 , training time : 10.591911554336548

Epoch : 67 , training time : 10.590144157409668

Epoch : 68 , training time : 10.600853204727173

Epoch : 69 , training time : 10.593989372253418

Epoch : 70 , training time : 10.605114459991455

Epoch : 71 , training time : 10.596534490585327

Epoch : 72 , training time : 10.596484899520874

Epoch : 73 , training time : 10.59665060043335

Epoch : 74 , training time : 10.601139783859253

Epoch : 75 , training time : 10.60957407951355

Epoch : 76 , training time : 10.611330270767212

Epoch : 77 , training time : 10.596938371658325

Epoch : 78 , training time : 10.611874341964722

Epoch : 79 , training time : 10.612704038619995

Epoch : 80 , training time : 10.616907835006714

Epoch : 81 , training time : 10.645366668701172

Epoch : 82 , training time : 10.615021467208862

Epoch : 83 , training time : 10.601182222366333

Epoch : 84 , training time : 10.598768711090088

Epoch : 85 , training time : 10.603297472000122

Epoch : 86 , training time : 10.611170530319214

Epoch : 87 , training time : 10.618456840515137

Epoch : 88 , training time : 10.603683710098267

Epoch : 89 , training time : 10.609476804733276

Epoch : 90 , training time : 10.597153902053833

Epoch : 91 , training time : 10.602348566055298

Epoch : 92 , training time : 10.627092838287354

Epoch : 93 , training time : 10.606618881225586

Epoch : 94 , training time : 10.648581504821777

Epoch : 95 , training time : 10.608660697937012

Epoch : 96 , training time : 10.602861166000366

Epoch : 97 , training time : 10.607563734054565

Epoch : 98 , training time : 10.611055612564087

Epoch : 99 , training time : 10.65071964263916

Epoch : 100 , training time : 10.602714538574219

---------------------Generated Image-----------------------------------------

A close up of a logo

Description automatically generated

---------------------Original Image-----------------------------------------

A close up of text on a white background

Description automatically generated

# LSTM for sequence problem recognition:

### LSTM:

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDS's (intrusion detection systems).

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

### Dataset: Absenteeism at Work (<https://archive.ics.uci.edu/ml/datasets/Absenteeism+at+work>)

The data set allows for several new combinations of attributes and attribute exclusions, or the modification of the attribute type (categorical, integer, or real) depending on the purpose of the research.The data set (Absenteeism at work - Part I) was used in academic research at the Universidade Nove de Julho - Postgraduate Program in Informatics and Knowledge Management.

### Source Code:

# -\*- coding: utf-8 -\*-

"""lstmRNNassignment.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1Z\_rsB5ZqJhl0TiuawaFJ3uLfq1GVRZ0q

"""

#use,if using kaggle

import os

os.chdir("../input")

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv("dataset.csv")

df = pd.DataFrame(data)

# print(df.columns.get\_loc("Social drinker"))

output = df[df.columns[df.columns.get\_loc("Social drinker")]]

inputs = df.drop(columns=['ID','Social drinker'])

xtrain, xtest, ytrain, ytest = train\_test\_split(

inputs, output, test\_size=0.33, random\_state=42)

print(xtrain.shape[1])

import keras

inputs = keras.layers.Input(shape = [xtrain.shape[1]])

x = keras.layers.Embedding(input\_dim=1000, output\_dim=50, input\_length=xtrain.shape[1]) (inputs)

x = keras.layers.LSTM(128) (x)

# x = keras.layers.Dropout(0.5)(x)

x = keras.layers.Dense(256, activation = 'relu')(x)

x = keras.layers.Dropout(0.5)(x)

x = keras.layers.Dense(1, activation = 'sigmoid')(x)

model = keras.models.Model(inputs,x)

model.compile(

loss='binary\_crossentropy',

optimizer=keras.optimizers.Adam(1e-4),

metrics=['accuracy']

)

model.summary()

history = model.fit(xtrain,ytrain, batch\_size=32, epochs = 10, verbose=1)

score = model.evaluate(xtest,ytest)

print("Accuracy for Testing Set: ",score[1]\*100, "%")

### Model Summary:

Model: "model\_12"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_12 (InputLayer) (None, 19) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_12 (Embedding) (None, 19, 50) 50000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_12 (LSTM) (None, 128) 91648

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_26 (Dense) (None, 256) 33024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_19 (Dropout) (None, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_27 (Dense) (None, 1) 257

=================================================================

Total params: 174,929

Trainable params: 174,929

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Result:

Epoch 1/10

495/495 [==============================] - 1s 2ms/step - loss: 0.6924 - accuracy: 0.5414

Epoch 2/10

495/495 [==============================] - 0s 747us/step - loss: 0.6891 - accuracy: 0.5980

Epoch 3/10

495/495 [==============================] - 0s 765us/step - loss: 0.6855 - accuracy: 0.5818

Epoch 4/10

495/495 [==============================] - 0s 721us/step - loss: 0.6785 - accuracy: 0.5919

Epoch 5/10

495/495 [==============================] - 0s 731us/step - loss: 0.6679 - accuracy: 0.5838

Epoch 6/10

495/495 [==============================] - 0s 710us/step - loss: 0.6455 - accuracy: 0.6141

Epoch 7/10

495/495 [==============================] - 0s 717us/step - loss: 0.5935 - accuracy: 0.6626

Epoch 8/10

495/495 [==============================] - 0s 686us/step - loss: 0.4974 - accuracy: 0.7616

Epoch 9/10

495/495 [==============================] - 0s 696us/step - loss: 0.4016 - accuracy: 0.9030

Epoch 10/10

495/495 [==============================] - 0s 684us/step - loss: 0.3123 - accuracy: 0.9374

245/245 [==============================] - 0s 526us/step

Accuracy for Testing Set: 93.06122660636902 %