Generating new MNIST digits with GAN

Source: https://github.com/Zackory/Keras-MNIST-GAN/blob/master/mnist_gan.py

In [2]:
```
import os
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt

from tensorflow import keras
from keras.layers import Input
from keras.models import Model, Sequential
from keras.layers.core import Reshape, Dense, Dropout, Flatten
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import Convolution2D, UpSampling2D
from keras.layers.normalization import BatchNormalization
from keras.datasets import mnist
from keras.optimizers import Adam
from keras import backend as K
from keras import initializers
```

In [3]:
```
# Deterministic output.
# Tired of seeing the same results every time? Remove the line below.
np.random.seed(1000)

# The results are a little better when the dimensionality of the random vector is only 10.
# The dimensionality has been left at 100 for consistency with other GAN implementations.
randomDim = 100
```

Step 1: Load the dataset

In [4]:
```
# Load MNIST data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = (X_train.astype(np.float32) - 127.5)/127.5
X_train = X_train.reshape(60000, 784)
```

In [5]:
```
# Optimizer
adam = Adam(lr=0.0002, beta_1=0.5)
```

Step 2: Build the generator model

In [8]:

```
generator = Sequential(name = 'generator')
generator.add(Dense(256, input_dim=randomDim, kernel_initializer=initializers.RandomNormal(stddev=0.02)))
generator.add(LeakyReLU(0.2))
generator.add(Dense(512))
generator.add(LeakyReLU(0.2))
generator.add(Dense(1024))
generator.add(LeakyReLU(0.2))
generator.add(Dense(784, activation='tanh'))
generator.compile(loss='binary_crossentropy', optimizer=adam)
generator.summary()
```

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 256)               25856
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dense_11 (Dense)             (None, 784)               803600
=================================================================
Total params: 1,486,352
Trainable params: 1,486,352
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Build the discriminator model

In [9]:

```
discriminator = Sequential(name = 'discriminator')
discriminator.add(Dense(1024, input_dim=784, kernel_initializer=initializers.RandomNormal(stddev=0.02)))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))
discriminator.add(Dense(512))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))
discriminator.add(Dense(256))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))
discriminator.add(Dense(1, activation='sigmoid'))
discriminator.compile(loss='binary_crossentropy', optimizer=adam)
discriminator.summary()
```

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 1024)              803840
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_13 (Dense)             (None, 512)               524800
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_14 (Dense)             (None, 256)               131328
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 256)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 257
=================================================================
Total params: 1,460,225
Trainable params: 1,460,225
Non-trainable params: 0
_________________________________________________________________
```

Step 4: Build GAN model = generator + discriminator

In [10]:

```
# Combined network
discriminator.trainable = False
ganInput = Input(shape=(randomDim,))
x = generator(ganInput)
ganOutput = discriminator(x)
gan = Model(inputs=ganInput, outputs=ganOutput)
gan.compile(loss='binary_crossentropy', optimizer=adam)
gan.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 100)]             0
_________________________________________________________________
generator (Sequential)      (None, 784)               1486352
_________________________________________________________________
discriminator (Sequential)  (None, 1)                 1460225
=================================================================
Total params: 2,946,577
Trainable params: 1,486,352
Non-trainable params: 1,460,225
_________________________________________________________________
```

Step 5: Prepare functions to produce and plot the results

In [25]:

```
dLosses = []
gLosses = []

# Plot the loss from each batch
def plotLoss(epoch):
    plt.figure(figsize=(10, 8))
    plt.plot(dLosses, label='Discriminitive loss')
    plt.plot(gLosses, label='Generative loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig('images/gan_loss_epoch_%d.png' % epoch)

# Create a wall of generated MNIST images
def plotGeneratedImages(epoch, examples=100, dim=(10, 10), figsize=(10, 10)):
    noise = np.random.normal(0, 1, size=[examples, randomDim])
    generatedImages = generator.predict(noise)
    generatedImages = generatedImages.reshape(examples, 28, 28)

    plt.figure(figsize=figsize)
    for i in range(generatedImages.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generatedImages[i], interpolation='nearest', cmap='gray_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('images/gan_generated_image_epoch_%d.png' % epoch)

# Save the generator and discriminator networks (and weights) for later use
def saveModels(epoch):
    generator.save('models/gan_generator_epoch_%d.h5' % epoch)
    discriminator.save('models/gan_discriminator_epoch_%d.h5' % epoch)

def train(epochs=1, batchSize=128):
    batchCount = X_train.shape[0] / batchSize
    print('Epochs:', epochs)
    print('Batch size:', batchSize)
    print('Batches per epoch:', batchCount)

    for e in range(1, epochs+1):
        print('-'*15, 'Epoch %d' % e, '-'*15)
        for i in tqdm(range(int(batchCount))):
            # Get a random set of input noise and images
            noise = np.random.normal(0, 1, size=[batchSize, randomDim])
            imageBatch = X_train[np.random.randint(0, X_train.shape[0], size=batchSize)]

            # Generate fake MNIST images
            generatedImages = generator.predict(noise)
            # print np.shape(imageBatch), np.shape(generatedImages)
            X = np.concatenate([imageBatch, generatedImages])

            # Labels for generated and real data
            yDis = np.zeros(2*batchSize)
            # One-sided label smoothing
            yDis[:batchSize] = 0.9

            # Train discriminator
            discriminator.trainable = True
            dloss = discriminator.train_on_batch(X, yDis)

            # Train generator
            noise = np.random.normal(0, 1, size=[batchSize, randomDim])
            yGen = np.ones(batchSize)
            discriminator.trainable = False
            gloss = gan.train_on_batch(noise, yGen)

        # Store loss of most recent batch from this epoch
        dLosses.append(dloss)
        gLosses.append(gloss)

        if e == 1 or e % 20 == 0:
            plotGeneratedImages(e)
            saveModels(e)

    # Plot losses from every epoch
    plotLoss(e)
```

Step 6: Fit GAN model & plot reults

In [27]:

```
train(100, 128)
```

```
100%|██████████| 468/468 [00:40<00:00, 11.46it/s]
  0%|          | 2/468 [00:00<00:40, 11.48it/s]

--------------- Epoch 99 ---------------

100%|██████████| 468/468 [00:39<00:00, 11.76it/s]
  0%|          | 2/468 [00:00<00:38, 12.24it/s]

--------------- Epoch 100 ---------------

100%|██████████| 468/468 [00:39<00:00, 11.99it/s]
```

In [ ]: