Import libraries

In [0]:

```
import keras
import tensorflow
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from keras.datasets import mnist
from keras import backend as K
import pickle
```

Load MNIST data

In [0]:

```
(train_x, _), (test_x, _) = mnist.load_data()
```

In [0]:

```
train_x = train_x.astype('float32')/255
test_x = test_x.astype('float32')/255
train_x = train_x.reshape(len(train_x), 28,28,1)
test_x = test_x.reshape(len(test_x), 28,28,1)
```

In [16]:

```
train_x.shape
```

Out[16]:

```
(60000, 28, 28, 1)
```

In [17]:

```
noise_factor = .99 * np.random.random_sample()
print(noise_factor)
```

```
0.8092934934850354
```

Introduce random uniform noise in the train and test set

- **Rather than taking a complete random uniform noise as the train set, random uniform noise is added to the train set.**
- **The random uniform noise is generated with mean and standard deviation as 1 and is kept between 0 to 1.**
- **If we use just random noise as the training data the model performs worse and hence some kind of grouping of the training data is needed for the model to train on.**

In [0]:

```
x_train_noisy = (train_x +  np.random.normal(loc=1.0, scale=1.0, size=train_x.shape))/ 2
x_test_noisy = (test_x + np.random.normal(loc=1.0, scale=1.0, size=test_x.shape)) / 2
```

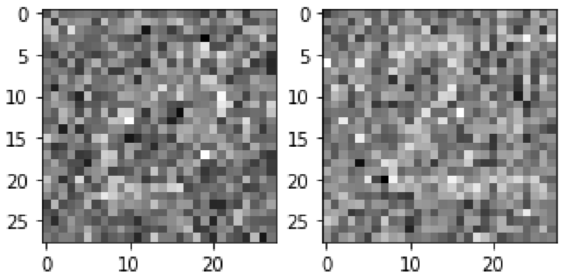
In [19]:

```
plt.figure(figsize=[5,5])
```

```python
# Display the first image in training data
plt.subplot(121)
curr_img = np.reshape(x_train_noisy[1], (28,28))
plt.imshow(curr_img, cmap='gray')

# Display the first image in testing data
plt.subplot(122)
curr_img = np.reshape(x_test_noisy[1], (28,28))
plt.imshow(curr_img, cmap='gray')
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x7fe6b0fe4b38>
```



Decoder part of the network

- **The assumption is that the trained model will be given a random noise vector of size 28 *28* 1 directly to the decoder part and is expected to output a MNIST like image**
- **Hence the decoder is given a 28 *28* 1 image and that is why no upsampling layer is there. If there is to be an encoded image that is being provided to the decoder after the encoder part, then the model needs to be adjusted accordingly and retrained.**

In [0]:

```python
input_img = Input(shape=(28, 28, 1))
```

In [0]:

```python
#x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
#x = MaxPooling2D((2, 2), padding='same')(x)
#x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
#x = MaxPooling2D((2, 2), padding='same')(x)
#x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
#encoded = MaxPooling2D((2, 2), padding='same')(x)
```

In [22]:

```python
#print(encoded)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(input_img)
print(x.shape)
#x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
print(x.shape)
#x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
print(x.shape)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
print(x.shape)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
print(x.shape)
#x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
print(decoded.shape)
```

```
(?, 28, 28, 8)
(?, 28, 28, 16)
(?, 28, 28, 32)
(?, 28, 28, 64)
```

```
(?, 28, 28, 64)
(?, 28, 28, 128)
(?, 28, 28, 1)
```

Define the model and compile with loss function

In [0]:

```
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

Training the model

- **The training is done for 200 epochs**
- **The train data is the noisy data generated earlier while the ground truth remains the same (original data).**

In [24]:

```
epochs = 200
autoencoder_train = autoencoder.fit(x_train_noisy, train_x, epochs=epochs, batch_size=12
8, shuffle=True, validation_data=(x_test_noisy, test_x),
                     verbose=2)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/200
 - 21s - loss: 0.2426 - val_loss: 0.1849
Epoch 2/200
 - 21s - loss: 0.1766 - val_loss: 0.1746
Epoch 3/200
 - 21s - loss: 0.1701 - val_loss: 0.1651
Epoch 4/200
 - 21s - loss: 0.1662 - val_loss: 0.1619
Epoch 5/200
 - 21s - loss: 0.1637 - val_loss: 0.1623
Epoch 6/200
 - 21s - loss: 0.1623 - val_loss: 0.1600
Epoch 7/200
 - 21s - loss: 0.1611 - val_loss: 0.1585
Epoch 8/200
 - 21s - loss: 0.1600 - val_loss: 0.1581
Epoch 9/200
 - 21s - loss: 0.1595 - val_loss: 0.1604
Epoch 10/200
 - 21s - loss: 0.1588 - val_loss: 0.1574
Epoch 11/200
 - 21s - loss: 0.1585 - val_loss: 0.1570
Epoch 12/200
 - 21s - loss: 0.1580 - val_loss: 0.1573
Epoch 13/200
 - 21s - loss: 0.1578 - val_loss: 0.1559
Epoch 14/200
 - 21s - loss: 0.1573 - val_loss: 0.1560
Epoch 15/200
 - 21s - loss: 0.1568 - val_loss: 0.1552
Epoch 16/200
 - 21s - loss: 0.1567 - val_loss: 0.1561
Epoch 17/200
 - 21s - loss: 0.1565 - val_loss: 0.1565
Epoch 18/200
 - 21s - loss: 0.1561 - val_loss: 0.1547
Epoch 19/200
 - 21s - loss: 0.1558 - val_loss: 0.1582
Epoch 20/200
 - 21s - loss: 0.1557 - val_loss: 0.1545
Epoch 21/200
 - 21s - loss: 0.1555 - val_loss: 0.1538
Epoch 22/200
```

```
 - 21s - loss: 0.1554 - val_loss: 0.1558
Epoch 23/200
 - 21s - loss: 0.1553 - val_loss: 0.1544
Epoch 24/200
 - 21s - loss: 0.1550 - val_loss: 0.1544
Epoch 25/200
 - 21s - loss: 0.1548 - val_loss: 0.1534
Epoch 26/200
 - 21s - loss: 0.1548 - val_loss: 0.1548
Epoch 27/200
 - 21s - loss: 0.1547 - val_loss: 0.1556
Epoch 28/200
 - 21s - loss: 0.1545 - val_loss: 0.1532
Epoch 29/200
 - 21s - loss: 0.1545 - val_loss: 0.1541
Epoch 30/200
 - 21s - loss: 0.1543 - val_loss: 0.1541
Epoch 31/200
 - 21s - loss: 0.1542 - val_loss: 0.1533
Epoch 32/200
 - 21s - loss: 0.1541 - val_loss: 0.1542
Epoch 33/200
 - 21s - loss: 0.1541 - val_loss: 0.1537
Epoch 34/200
 - 21s - loss: 0.1539 - val_loss: 0.1528
Epoch 35/200
 - 21s - loss: 0.1537 - val_loss: 0.1530
Epoch 36/200
 - 21s - loss: 0.1537 - val_loss: 0.1531
Epoch 37/200
 - 21s - loss: 0.1537 - val_loss: 0.1532
Epoch 38/200
 - 21s - loss: 0.1536 - val_loss: 0.1541
Epoch 39/200
 - 21s - loss: 0.1534 - val_loss: 0.1523
Epoch 40/200
 - 21s - loss: 0.1533 - val_loss: 0.1524
Epoch 41/200
 - 21s - loss: 0.1533 - val_loss: 0.1529
Epoch 42/200
 - 21s - loss: 0.1532 - val_loss: 0.1525
Epoch 43/200
 - 21s - loss: 0.1532 - val_loss: 0.1523
Epoch 44/200
 - 21s - loss: 0.1531 - val_loss: 0.1534
Epoch 45/200
 - 21s - loss: 0.1531 - val_loss: 0.1523
Epoch 46/200
 - 21s - loss: 0.1530 - val_loss: 0.1519
Epoch 47/200
 - 21s - loss: 0.1530 - val_loss: 0.1526
Epoch 48/200
 - 21s - loss: 0.1529 - val_loss: 0.1524
Epoch 49/200
 - 21s - loss: 0.1529 - val_loss: 0.1536
Epoch 50/200
 - 21s - loss: 0.1528 - val_loss: 0.1520
Epoch 51/200
 - 21s - loss: 0.1528 - val_loss: 0.1530
Epoch 52/200
 - 21s - loss: 0.1527 - val_loss: 0.1520
Epoch 53/200
 - 21s - loss: 0.1526 - val_loss: 0.1518
Epoch 54/200
 - 21s - loss: 0.1526 - val_loss: 0.1517
Epoch 55/200
 - 21s - loss: 0.1525 - val_loss: 0.1523
Epoch 56/200
 - 21s - loss: 0.1525 - val_loss: 0.1534
Epoch 57/200
 - 21s - loss: 0.1524 - val_loss: 0.1516
```

```
 - 21s - loss: 0.1524 - val_loss: 0.1516
Epoch 58/200
 - 21s - loss: 0.1523 - val_loss: 0.1515
Epoch 59/200
 - 21s - loss: 0.1523 - val_loss: 0.1517
Epoch 60/200
 - 21s - loss: 0.1523 - val_loss: 0.1516
Epoch 61/200
 - 21s - loss: 0.1523 - val_loss: 0.1518
Epoch 62/200
 - 21s - loss: 0.1523 - val_loss: 0.1517
Epoch 63/200
 - 21s - loss: 0.1523 - val_loss: 0.1518
Epoch 64/200
 - 21s - loss: 0.1522 - val_loss: 0.1515
Epoch 65/200
 - 21s - loss: 0.1521 - val_loss: 0.1513
Epoch 66/200
 - 21s - loss: 0.1520 - val_loss: 0.1520
Epoch 67/200
 - 21s - loss: 0.1520 - val_loss: 0.1514
Epoch 68/200
 - 21s - loss: 0.1520 - val_loss: 0.1518
Epoch 69/200
 - 21s - loss: 0.1520 - val_loss: 0.1520
Epoch 70/200
 - 21s - loss: 0.1519 - val_loss: 0.1513
Epoch 71/200
 - 21s - loss: 0.1520 - val_loss: 0.1515
Epoch 72/200
 - 21s - loss: 0.1519 - val_loss: 0.1511
Epoch 73/200
 - 21s - loss: 0.1518 - val_loss: 0.1539
Epoch 74/200
 - 21s - loss: 0.1519 - val_loss: 0.1512
Epoch 75/200
 - 21s - loss: 0.1519 - val_loss: 0.1510
Epoch 76/200
 - 21s - loss: 0.1517 - val_loss: 0.1529
Epoch 77/200
 - 21s - loss: 0.1517 - val_loss: 0.1510
Epoch 78/200
 - 21s - loss: 0.1516 - val_loss: 0.1511
Epoch 79/200
 - 21s - loss: 0.1516 - val_loss: 0.1510
Epoch 80/200
 - 21s - loss: 0.1516 - val_loss: 0.1509
Epoch 81/200
 - 21s - loss: 0.1516 - val_loss: 0.1512
Epoch 82/200
 - 21s - loss: 0.1517 - val_loss: 0.1512
Epoch 83/200
 - 21s - loss: 0.1515 - val_loss: 0.1519
Epoch 84/200
 - 21s - loss: 0.1515 - val_loss: 0.1509
Epoch 85/200
 - 21s - loss: 0.1515 - val_loss: 0.1520
Epoch 86/200
 - 21s - loss: 0.1516 - val_loss: 0.1522
Epoch 87/200
 - 21s - loss: 0.1515 - val_loss: 0.1523
Epoch 88/200
 - 21s - loss: 0.1515 - val_loss: 0.1512
Epoch 89/200
 - 21s - loss: 0.1514 - val_loss: 0.1509
Epoch 90/200
 - 21s - loss: 0.1514 - val_loss: 0.1509
Epoch 91/200
 - 21s - loss: 0.1513 - val_loss: 0.1514
Epoch 92/200
 - 21s - loss: 0.1513 - val_loss: 0.1519
```

```
Epoch 93/200
 - 21s - loss: 0.1513 - val_loss: 0.1509
Epoch 94/200
 - 21s - loss: 0.1513 - val_loss: 0.1510
Epoch 95/200
 - 21s - loss: 0.1513 - val_loss: 0.1509
Epoch 96/200
 - 21s - loss: 0.1513 - val_loss: 0.1510
Epoch 97/200
 - 21s - loss: 0.1513 - val_loss: 0.1508
Epoch 98/200
 - 21s - loss: 0.1512 - val_loss: 0.1508
Epoch 99/200
 - 21s - loss: 0.1513 - val_loss: 0.1508
Epoch 100/200
 - 21s - loss: 0.1512 - val_loss: 0.1508
Epoch 101/200
 - 21s - loss: 0.1512 - val_loss: 0.1517
Epoch 102/200
 - 21s - loss: 0.1512 - val_loss: 0.1515
Epoch 103/200
 - 21s - loss: 0.1511 - val_loss: 0.1514
Epoch 104/200
 - 21s - loss: 0.1511 - val_loss: 0.1518
Epoch 105/200
 - 21s - loss: 0.1512 - val_loss: 0.1515
Epoch 106/200
 - 21s - loss: 0.1511 - val_loss: 0.1508
Epoch 107/200
 - 21s - loss: 0.1511 - val_loss: 0.1509
Epoch 108/200
 - 21s - loss: 0.1511 - val_loss: 0.1509
Epoch 109/200
 - 21s - loss: 0.1510 - val_loss: 0.1519
Epoch 110/200
 - 21s - loss: 0.1510 - val_loss: 0.1520
Epoch 111/200
 - 21s - loss: 0.1510 - val_loss: 0.1509
Epoch 112/200
 - 21s - loss: 0.1509 - val_loss: 0.1509
Epoch 113/200
 - 21s - loss: 0.1510 - val_loss: 0.1507
Epoch 114/200
 - 21s - loss: 0.1510 - val_loss: 0.1521
Epoch 115/200
 - 21s - loss: 0.1511 - val_loss: 0.1507
Epoch 116/200
 - 21s - loss: 0.1509 - val_loss: 0.1517
Epoch 117/200
 - 21s - loss: 0.1509 - val_loss: 0.1512
Epoch 118/200
 - 21s - loss: 0.1509 - val_loss: 0.1513
Epoch 119/200
 - 21s - loss: 0.1508 - val_loss: 0.1509
Epoch 120/200
 - 21s - loss: 0.1509 - val_loss: 0.1525
Epoch 121/200
 - 21s - loss: 0.1508 - val_loss: 0.1508
Epoch 122/200
 - 21s - loss: 0.1509 - val_loss: 0.1507
Epoch 123/200
 - 21s - loss: 0.1509 - val_loss: 0.1505
Epoch 124/200
 - 21s - loss: 0.1509 - val_loss: 0.1511
Epoch 125/200
 - 21s - loss: 0.1508 - val_loss: 0.1506
Epoch 126/200
 - 21s - loss: 0.1508 - val_loss: 0.1507
Epoch 127/200
 - 21s - loss: 0.1507 - val_loss: 0.1504
Epoch 128/200
```

```
Epoch 128/200
 - 21s - loss: 0.1507 - val_loss: 0.1511
Epoch 129/200
 - 21s - loss: 0.1507 - val_loss: 0.1505
Epoch 130/200
 - 21s - loss: 0.1507 - val_loss: 0.1507
Epoch 131/200
 - 21s - loss: 0.1507 - val_loss: 0.1508
Epoch 132/200
 - 21s - loss: 0.1506 - val_loss: 0.1508
Epoch 133/200
 - 21s - loss: 0.1507 - val_loss: 0.1508
Epoch 134/200
 - 21s - loss: 0.1507 - val_loss: 0.1508
Epoch 135/200
 - 21s - loss: 0.1507 - val_loss: 0.1505
Epoch 136/200
 - 21s - loss: 0.1507 - val_loss: 0.1507
Epoch 137/200
 - 21s - loss: 0.1506 - val_loss: 0.1505
Epoch 138/200
 - 21s - loss: 0.1506 - val_loss: 0.1515
Epoch 139/200
 - 21s - loss: 0.1506 - val_loss: 0.1506
Epoch 140/200
 - 21s - loss: 0.1506 - val_loss: 0.1506
Epoch 141/200
 - 21s - loss: 0.1506 - val_loss: 0.1505
Epoch 142/200
 - 21s - loss: 0.1506 - val_loss: 0.1507
Epoch 143/200
 - 21s - loss: 0.1506 - val_loss: 0.1508
Epoch 144/200
 - 21s - loss: 0.1506 - val_loss: 0.1513
Epoch 145/200
 - 21s - loss: 0.1506 - val_loss: 0.1508
Epoch 146/200
 - 21s - loss: 0.1506 - val_loss: 0.1519
Epoch 147/200
 - 21s - loss: 0.1505 - val_loss: 0.1512
Epoch 148/200
 - 21s - loss: 0.1506 - val_loss: 0.1508
Epoch 149/200
 - 21s - loss: 0.1504 - val_loss: 0.1511
Epoch 150/200
 - 21s - loss: 0.1504 - val_loss: 0.1506
Epoch 151/200
 - 21s - loss: 0.1505 - val_loss: 0.1512
Epoch 152/200
 - 21s - loss: 0.1504 - val_loss: 0.1505
Epoch 153/200
 - 21s - loss: 0.1505 - val_loss: 0.1507
Epoch 154/200
 - 21s - loss: 0.1504 - val_loss: 0.1507
Epoch 155/200
 - 21s - loss: 0.1504 - val_loss: 0.1506
Epoch 156/200
 - 21s - loss: 0.1503 - val_loss: 0.1511
Epoch 157/200
 - 21s - loss: 0.1504 - val_loss: 0.1506
Epoch 158/200
 - 21s - loss: 0.1504 - val_loss: 0.1509
Epoch 159/200
 - 21s - loss: 0.1503 - val_loss: 0.1512
Epoch 160/200
 - 21s - loss: 0.1504 - val_loss: 0.1504
Epoch 161/200
 - 21s - loss: 0.1503 - val_loss: 0.1505
Epoch 162/200
 - 21s - loss: 0.1503 - val_loss: 0.1509
Epoch 163/200
```

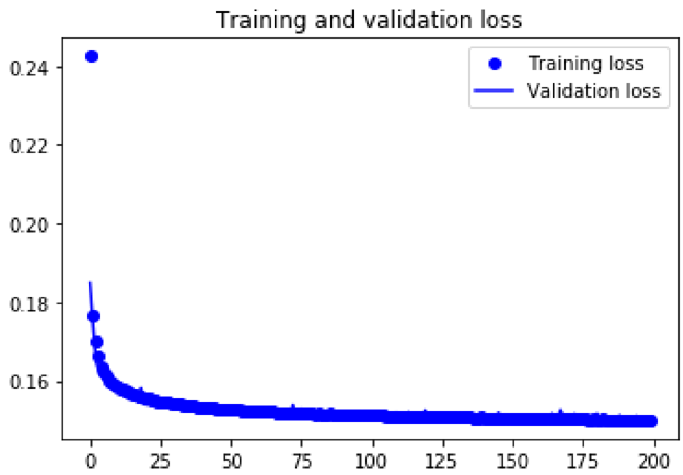
```
 - 21s - loss: 0.1503 - val_loss: 0.1506
Epoch 164/200
 - 21s - loss: 0.1504 - val_loss: 0.1505
Epoch 165/200
 - 21s - loss: 0.1503 - val_loss: 0.1517
Epoch 166/200
 - 21s - loss: 0.1503 - val_loss: 0.1503
Epoch 167/200
 - 21s - loss: 0.1502 - val_loss: 0.1506
Epoch 168/200
 - 21s - loss: 0.1503 - val_loss: 0.1526
Epoch 169/200
 - 21s - loss: 0.1502 - val_loss: 0.1511
Epoch 170/200
 - 21s - loss: 0.1503 - val_loss: 0.1505
Epoch 171/200
 - 21s - loss: 0.1502 - val_loss: 0.1505
Epoch 172/200
 - 21s - loss: 0.1502 - val_loss: 0.1505
Epoch 173/200
 - 21s - loss: 0.1502 - val_loss: 0.1516
Epoch 174/200
 - 21s - loss: 0.1502 - val_loss: 0.1506
Epoch 175/200
 - 21s - loss: 0.1502 - val_loss: 0.1505
Epoch 176/200
 - 21s - loss: 0.1502 - val_loss: 0.1505
Epoch 177/200
 - 21s - loss: 0.1502 - val_loss: 0.1508
Epoch 178/200
 - 21s - loss: 0.1501 - val_loss: 0.1504
Epoch 179/200
 - 21s - loss: 0.1502 - val_loss: 0.1506
Epoch 180/200
 - 21s - loss: 0.1502 - val_loss: 0.1507
Epoch 181/200
 - 21s - loss: 0.1501 - val_loss: 0.1517
Epoch 182/200
 - 21s - loss: 0.1501 - val_loss: 0.1507
Epoch 183/200
 - 21s - loss: 0.1502 - val_loss: 0.1507
Epoch 184/200
 - 21s - loss: 0.1501 - val_loss: 0.1511
Epoch 185/200
 - 21s - loss: 0.1501 - val_loss: 0.1507
Epoch 186/200
 - 21s - loss: 0.1501 - val_loss: 0.1508
Epoch 187/200
 - 21s - loss: 0.1501 - val_loss: 0.1505
Epoch 188/200
 - 21s - loss: 0.1500 - val_loss: 0.1508
Epoch 189/200
 - 21s - loss: 0.1501 - val_loss: 0.1512
Epoch 190/200
 - 21s - loss: 0.1501 - val_loss: 0.1507
Epoch 191/200
 - 21s - loss: 0.1500 - val_loss: 0.1504
Epoch 192/200
 - 21s - loss: 0.1500 - val_loss: 0.1503
Epoch 193/200
 - 21s - loss: 0.1500 - val_loss: 0.1509
Epoch 194/200
 - 21s - loss: 0.1500 - val_loss: 0.1503
Epoch 195/200
 - 21s - loss: 0.1501 - val_loss: 0.1511
Epoch 196/200
 - 21s - loss: 0.1501 - val_loss: 0.1505
Epoch 197/200
 - 21s - loss: 0.1500 - val_loss: 0.1508
Epoch 198/200
 - 21s - loss: 0.1500 - val_loss: 0.1504
```

```
 - 21s - loss: 0.1500 - val_loss: 0.1504
Epoch 199/200
 - 21s - loss: 0.1500 - val_loss: 0.1505
Epoch 200/200
 - 21s - loss: 0.1500 - val_loss: 0.1508
```

Plot the curve between training and validation loss

In [25]:

```
loss = autoencoder_train.history['loss']
val_loss = autoencoder_train.history['val_loss']
epoch = range(epochs)
plt.figure()
plt.plot(epoch, loss, 'bo', label='Training loss')
plt.plot(epoch, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



Predict the decoded images from the test set

In [0]:

```
decoded_imgs = autoencoder.predict(x_test_noisy)
```

Plot the decoded images against the original (without noise) images

In [27]:

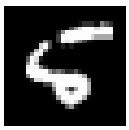
```
n = 10
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    #plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.imshow(test_x[i].reshape(28, 28))
    plt.gray()
    ax.set_axis_off()


    # display reconstruction
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.set_axis_off()

plt.show()
```

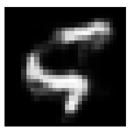
In [0]:

```
def show_img(img):
  n,i = 10,0
  plt.figure(figsize=(10, 4), dpi=100)

  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(img.reshape(28, 28))
  plt.gray()
  ax.set_axis_off()

  plt.show()
```

In [29]:

```
i = 8
t = x_test_noisy[i]
t.shape
x = np.expand_dims(t, axis=0)
x.shape
d = autoencoder.predict(x)
show_img(test_x[i])
show_img(d)
```





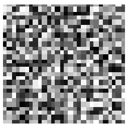
Generate a random noise vector and output as a MNIST like image

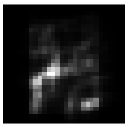
In [30]:

```
# random vector to mnist image

r = np.random.randint(0, 255, (28,28,1))
print((r==t).all())
r = np.expand_dims(r, axis=0)
r = r.astype('float32')/255
r = r.reshape(len(r), 28,28,1)
r_dec = autoencoder.predict(r)
```

```
show_img(r)
show_img(r_dec)
```

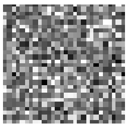
False

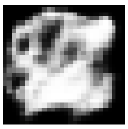




In [31]:

```
# replacing values at random indexes of a noisy image and then generating back the origin
al
y = t
# random boolean mask for which values will be changed
mask = np.random.randint(0,2,size=y.shape).astype(np.bool)

# random matrix the same shape of your data
r = np.random.rand(*y.shape)*np.max(y)

# use your mask to replace values in your input array
y[mask] = r[mask]

y = np.expand_dims(y, axis=0)
y_dec = autoencoder.predict(y)
show_img(t)
show_img(y_dec)
```





In [0]:

```

```