cse291_hw2_denoising_autoencoder

February 16, 2018

```
In [1]: %matplotlib inline
        import matplotlib
        import matplotlib.pyplot as plt
        import tensorflow as tf
        import numpy as np
        import time
        import sklearn
        from tensorflow.examples.tutorials.mnist import input_data
        from sklearn.metrics import euclidean_distances
```

```
In [2]: #Read Data
        mnist = input_data.read_data_sets(".",one_hot = True)
```

```
Extracting .\train-images-idx3-ubyte.gz
Extracting .\train-labels-idx1-ubyte.gz
Extracting .\t10k-images-idx3-ubyte.gz
Extracting .\t10k-labels-idx1-ubyte.gz
```

```
In [3]: # def TRAIN_SIZE(num):
        #     print ('Total Training Images in Dataset = ' + str(mnist.train.images.shape))
        #     print ('--------------------------------------------------')
        #     x_train = mnist.train.images[:num,:]
        #     print ('x_train Examples Loaded = ' + str(x_train.shape))
        #     y_train = mnist.train.labels[:num,:]
        #     print ('y_train Examples Loaded = ' + str(y_train.shape))
        #     print('')
        #     return x_train, y_train

        def plot_image(img):
            plt.imshow(img.reshape(28,28),cmap="binary")

        def gaussian_noise(img, sigma):
            mean = 0
            noisy_img = img + 0.5*np.random.normal(mean, sigma, img.shape)
            return noisy_img
```

```
In [4]: def encoder_layer(e):
            conv1 = tf.layers.conv2d(e, filters=32, kernel_size=(3,3), strides=(1,1), \
```

```
                              padding='same', activation=tf.nn.relu, name='conv1')
    pool1 = tf.layers.max_pooling2d(conv1, pool_size=(2,2), strides=(2,2), name='pool1')
    conv2 = tf.layers.conv2d(pool1, filters=16, kernel_size=(3,3), strides=(1,1), \
                             padding='same', activation=tf.nn.relu, name='conv2')
    pool2 = tf.layers.max_pooling2d(conv2, pool_size=(2,2), strides=(2,2), name='pool2')
    flat = tf.reshape(pool2, [-1, 7*7*16])
    fc1 = tf.layers.dense(inputs=flat, units=1000, activation=tf.nn.relu)
    fc2 = tf.layers.dense(inputs=fc1, units=100, activation=tf.nn.relu)
    return fc2

def decoder_layer(d):
    enc = tf.reshape(d, [-1,10,10,1])
#     deconv1 = tf.layers.conv2d_transpose(enc, filters=32, kernel_size=(5,5), strides=(
#                                          padding='valid', activation=tf.nn.relu, name=
#         deconv2 = tf.layers.conv2d_transpose(deconv1, filters=1, kernel_size=(3,3), st
#                                          padding='same', name='deconv2')
    deconv1 = tf.layers.conv2d_transpose(enc, filters=32, kernel_size=(5,5), strides=(1,
                                         padding='valid', activation=tf.nn.relu, name='d

    deconv2 = tf.layers.conv2d_transpose(deconv1, filters=1, kernel_size=(3,3), strides=
                                         padding='same', name='deconv2')
    return deconv2
```

In [5]:
```
#Placeholder for original noise free images
TARGET = tf.placeholder(tf.float32, [None, 784])
#Reshape original images
target = tf.reshape(TARGET, [-1, 28, 28, 1])

#Placeholder for noisy input images
X = tf.placeholder(tf.float32, [None, 784])
#Reshape noisy images
x = tf.reshape(X, [-1, 28, 28, 1])

encode = encoder_layer(x)
decode = decoder_layer(encode)

# reshape reconstructed image
RECONSTRUCTED = tf.reshape(decode, [-1,784])

loss = tf.reduce_mean(tf.square(TARGET-RECONSTRUCTED))
opt = tf.train.AdamOptimizer(learning_rate=0.009).minimize(loss)
```

In [6]:
```
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

#Make batches to train
epochs = 1
batch_size = 128
```

```python
loss_val = []

for epoch in range(epochs):
    total_loss = 0.0
    for i in range(mnist.train.num_examples//batch_size):
        orig, _ = mnist.train.next_batch(batch_size)
        noisy = gaussian_noise(orig, 0.1)
        _, L, R = sess.run([opt, loss, RECONSTRUCTED], feed_dict = {TARGET: orig, X: noi
        loss_val.append(L)
        print(i,': ',L)
        total_loss += L
    print('Epoch {} - Total loss: {}'.format(epoch+1, total_loss))
```

```
0 :  0.114301525
1 :  0.11273749
2 :  0.101647675
3 :  0.096594624
4 :  0.09525443
5 :  0.082287766
6 :  0.07695998
7 :  0.08388837
8 :  0.07779556
9 :  0.073846556
10 :  0.07389357
11 :  0.077573165
12 :  0.07648513
13 :  0.073966116
14 :  0.07251046
15 :  0.07197156
16 :  0.07383
17 :  0.07106408
18 :  0.06916287
19 :  0.068117656
20 :  0.06462246
21 :  0.067905605
22 :  0.06902473
23 :  0.06766471
24 :  0.06893855
25 :  0.06464924
26 :  0.0678722
27 :  0.06481902
28 :  0.06521715
29 :  0.067879826
30 :  0.06788912
31 :  0.06223636
32 :  0.06555415
33 :  0.06121555
34 :  0.06274305
```

35 : 0.058242396
36 : 0.061655506
37 : 0.061903637
38 : 0.05885388
39 : 0.06278045
40 : 0.060323473
41 : 0.06128627
42 : 0.056939702
43 : 0.057447396
44 : 0.05625437
45 : 0.057269737
46 : 0.055517398
47 : 0.056551315
48 : 0.054190338
49 : 0.05874743
50 : 0.054096196
51 : 0.054598995
52 : 0.054555386
53 : 0.050006855
54 : 0.05573545
55 : 0.051953826
56 : 0.055667564
57 : 0.05256717
58 : 0.052390702
59 : 0.051852502
60 : 0.050994694
61 : 0.054410223
62 : 0.053344022
63 : 0.05529183
64 : 0.055596605
65 : 0.051399235
66 : 0.050881308
67 : 0.05061131
68 : 0.052790582
69 : 0.048366707
70 : 0.05058883
71 : 0.04900838
72 : 0.049768366
73 : 0.049468268
74 : 0.04926403
75 : 0.053204663
76 : 0.047648937
77 : 0.049694076
78 : 0.04968254
79 : 0.04868665
80 : 0.049002536
81 : 0.04863791
82 : 0.04718827

83 : 0.04731841
84 : 0.047496043
85 : 0.047192562
86 : 0.046093084
87 : 0.047736976
88 : 0.047600824
89 : 0.04727902
90 : 0.0464917
91 : 0.045165207
92 : 0.04644295
93 : 0.04621889
94 : 0.04489754
95 : 0.046612274
96 : 0.045313448
97 : 0.047324512
98 : 0.04349876
99 : 0.046389114
100 : 0.046701293
101 : 0.04384357
102 : 0.04736683
103 : 0.04562454
104 : 0.047365822
105 : 0.047203664
106 : 0.04347652
107 : 0.046174627
108 : 0.05039946
109 : 0.045960527
110 : 0.05080856
111 : 0.04563069
112 : 0.04453954
113 : 0.042771734
114 : 0.045367755
115 : 0.046368744
116 : 0.045534868
117 : 0.043660212
118 : 0.042989593
119 : 0.044331785
120 : 0.04104963
121 : 0.04352636
122 : 0.04191214
123 : 0.042419277
124 : 0.043397263
125 : 0.043238074
126 : 0.04233298
127 : 0.044360705
128 : 0.04068767
129 : 0.043321192
130 : 0.040242452

131 : 0.041154515
132 : 0.040423024
133 : 0.041487187
134 : 0.041454226
135 : 0.03889644
136 : 0.041417282
137 : 0.041526433
138 : 0.04016429
139 : 0.03995727
140 : 0.042218085
141 : 0.041367766
142 : 0.042474385
143 : 0.04066622
144 : 0.04071346
145 : 0.041584443
146 : 0.041028887
147 : 0.040426347
148 : 0.039887544
149 : 0.040650807
150 : 0.041113287
151 : 0.040157292
152 : 0.0409016
153 : 0.042181667
154 : 0.03960948
155 : 0.039194185
156 : 0.037633106
157 : 0.038372446
158 : 0.040080868
159 : 0.040862147
160 : 0.04034986
161 : 0.037575655
162 : 0.036410533
163 : 0.038051333
164 : 0.03782411
165 : 0.036997907
166 : 0.039128143
167 : 0.037659347
168 : 0.037646633
169 : 0.037019745
170 : 0.039886598
171 : 0.0398704
172 : 0.038512215
173 : 0.039344337
174 : 0.03600647
175 : 0.034767617
176 : 0.03818278
177 : 0.037454378
178 : 0.036545154

179 : 0.038528763
180 : 0.03729623
181 : 0.03666106
182 : 0.03588311
183 : 0.0368945
184 : 0.038730048
185 : 0.038809244
186 : 0.036844093
187 : 0.036583047
188 : 0.04040232
189 : 0.036507394
190 : 0.03666804
191 : 0.03868423
192 : 0.036966287
193 : 0.03651273
194 : 0.035240676
195 : 0.03697382
196 : 0.038656875
197 : 0.037801128
198 : 0.03863099
199 : 0.03732403
200 : 0.033750314
201 : 0.039156448
202 : 0.034827486
203 : 0.035969082
204 : 0.035084277
205 : 0.037397485
206 : 0.03754862
207 : 0.038906768
208 : 0.03567562
209 : 0.03625929
210 : 0.03593713
211 : 0.03708166
212 : 0.03629377
213 : 0.036202714
214 : 0.036296945
215 : 0.036939174
216 : 0.035734035
217 : 0.03667836
218 : 0.03374225
219 : 0.033939585
220 : 0.03535691
221 : 0.035705723
222 : 0.03526696
223 : 0.035512023
224 : 0.034676425
225 : 0.036059983
226 : 0.035242926

227 : 0.034987785
228 : 0.036461342
229 : 0.03598322
230 : 0.03755896
231 : 0.038593926
232 : 0.034671463
233 : 0.033641946
234 : 0.03412273
235 : 0.038062632
236 : 0.03451236
237 : 0.03766378
238 : 0.042815607
239 : 0.03595974
240 : 0.041738003
241 : 0.046679482
242 : 0.04033051
243 : 0.049477264
244 : 0.036980096
245 : 0.041398704
246 : 0.03586861
247 : 0.04540748
248 : 0.039099682
249 : 0.04381601
250 : 0.038222425
251 : 0.039198093
252 : 0.03427107
253 : 0.038663335
254 : 0.03934901
255 : 0.037554424
256 : 0.038198203
257 : 0.03618698
258 : 0.039702974
259 : 0.03647502
260 : 0.037600122
261 : 0.03689404
262 : 0.036466442
263 : 0.036812197
264 : 0.034487963
265 : 0.031399876
266 : 0.035693467
267 : 0.03540497
268 : 0.03384561
269 : 0.03532978
270 : 0.034436245
271 : 0.03385922
272 : 0.036183815
273 : 0.03360635
274 : 0.034619715

275 : 0.037874624
276 : 0.035909075
277 : 0.036770318
278 : 0.03435082
279 : 0.03442901
280 : 0.035752237
281 : 0.03226417
282 : 0.0340466
283 : 0.032286666
284 : 0.034798115
285 : 0.032374714
286 : 0.0345856
287 : 0.032878958
288 : 0.035739075
289 : 0.033876996
290 : 0.035788648
291 : 0.034103133
292 : 0.032126125
293 : 0.032419253
294 : 0.033593398
295 : 0.03319531
296 : 0.0334205
297 : 0.033828687
298 : 0.034525216
299 : 0.03443701
300 : 0.035945453
301 : 0.03411845
302 : 0.035791844
303 : 0.034933753
304 : 0.032162517
305 : 0.033129327
306 : 0.032504212
307 : 0.033231247
308 : 0.033969346
309 : 0.032807205
310 : 0.03335113
311 : 0.033722978
312 : 0.032173406
313 : 0.034013275
314 : 0.03366427
315 : 0.033358388
316 : 0.03158832
317 : 0.03324277
318 : 0.03377102
319 : 0.032940894
320 : 0.031615812
321 : 0.0333478
322 : 0.033842966

```
323 :  0.0334084
324 :  0.033721775
325 :  0.03270568
326 :  0.033354808
327 :  0.033943325
328 :  0.035066158
329 :  0.033981387
330 :  0.0336314
331 :  0.032774862
332 :  0.031517643
333 :  0.03535733
334 :  0.03242543
335 :  0.031893898
336 :  0.031999435
337 :  0.03264354
338 :  0.032201968
339 :  0.03303245
340 :  0.034346834
341 :  0.032528702
342 :  0.03352374
343 :  0.031356927
344 :  0.033661887
345 :  0.033341736
346 :  0.031894617
347 :  0.034146816
348 :  0.034218322
349 :  0.032682642
350 :  0.030225279
351 :  0.033814818
352 :  0.033617403
353 :  0.034255184
354 :  0.032541983
355 :  0.034526065
356 :  0.033068433
357 :  0.033014737
358 :  0.03219421
359 :  0.031223193
360 :  0.032703288
361 :  0.0319854
362 :  0.0329243
363 :  0.032855637
364 :  0.03220692
365 :  0.031973887
366 :  0.033450384
367 :  0.033309154
368 :  0.03241909
369 :  0.033612605
370 :  0.03271362
```

371 : 0.032780755
372 : 0.033428352
373 : 0.03258263
374 : 0.032575637
375 : 0.03222521
376 : 0.031328723
377 : 0.03255429
378 : 0.031633522
379 : 0.032997753
380 : 0.033373885
381 : 0.03171907
382 : 0.030126683
383 : 0.034054566
384 : 0.03307959
385 : 0.031976376
386 : 0.032655403
387 : 0.03235165
388 : 0.030789142
389 : 0.03302019
390 : 0.031997778
391 : 0.031899437
392 : 0.03218782
393 : 0.03213812
394 : 0.032819062
395 : 0.032368015
396 : 0.03184752
397 : 0.031076353
398 : 0.031180194
399 : 0.032188255
400 : 0.03205135
401 : 0.031272415
402 : 0.031246493
403 : 0.031140924
404 : 0.032461464
405 : 0.032407377
406 : 0.030477531
407 : 0.031411987
408 : 0.031177744
409 : 0.0319795
410 : 0.03181419
411 : 0.030953582
412 : 0.03269819
413 : 0.032109357
414 : 0.03156284
415 : 0.03207601
416 : 0.032088295
417 : 0.032330003
418 : 0.033526868

```
419 :  0.03310865
420 :  0.033171892
421 :  0.030739402
422 :  0.03167176
423 :  0.03323218
424 :  0.032205094
425 :  0.031271532
426 :  0.03309433
427 :  0.0348631
428 :  0.03408198
Epoch 1 - Total loss: 17.911468723788857
```

In [9]:
```python
# Examples to test reconstruction
n_examples = 10

test_orig, _ = mnist.test.next_batch(n_examples)
test_noisy = gaussian_noise(test_orig, 0.1)
recon = sess.run(RECONSTRUCTED, feed_dict = {TARGET: test_orig, X: test_noisy})
# print(recon.shape)

# Visualize original, noisy and reconstructed images
#      Row 1 : Original
#      Row 2 : Noisy
#      Row 3 : Reconstructed
fig, axs = plt.subplots(3, n_examples, figsize=(15,4))
for example_i in range(n_examples):
    axs[0][example_i].imshow(np.reshape(test_orig[example_i,:], (28, 28)))
    axs[1][example_i].imshow(np.reshape(test_noisy[example_i,:], (28, 28)))
    axs[2][example_i].imshow(np.reshape(recon[example_i, :], (28, 28)))
```