RNN

2019年5月1日 19:52

In [1]:
```
# As usual, a bit of setup
import time, os, json
import numpy as np
import matplotlib.pyplot as plt

from cs231n.gradient_check import eval_numerical_gradient, eval_numerical_gradient_array
from cs231n.rnn_layers import *
from cs231n.captioning_solver import CaptioningSolver
from cs231n.classifiers.rnn import CaptioningRNN
from cs231n.coco_utils import load_coco_data, sample_coco_minibatch, decode_captions
from cs231n.image_utils import image_from_url

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
    """ returns relative error """
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

这个作业用的是coco数据集，如果是windows的话，做作业之前先在assignment3的cs231n的datasets里面打开记事本get_coco_captioning，直接复制里面的网址就可以下载数据集了，下下来之后解压就好了。

In [3]:
```
!pip install h5py
```

```
Looking in indexes: http://mirrors.aliyun.com/pypi/simple/
Requirement already satisfied: h5py in d:\anaconda\envs\cs231n\lib\site-packages (2.9.0)
Requirement already satisfied: numpy>=1.7 in d:\anaconda\envs\cs231n\lib\site-packages (from h5py) (1.16.2)
Requirement already satisfied: six in d:\anaconda\envs\cs231n\lib\site-packages (from h5py) (1.12.0)
```

pip安装h5py包

In [2]:
```
# Load COCO data from disk; this returns a dictionary
# We'll work with dimensionality-reduced features for this notebook, but feel
# free to experiment with the original features by changing the flag below.
data = load_coco_data(pca_features=True)

# Print out all the keys and values from the data dictionary
for k, v in data.items():
    if type(v) == np.ndarray:
        print(k, type(v), v.shape, v.dtype)
    else:
        print(k, type(v), len(v))
```

```
train_captions <class 'numpy.ndarray'> (400135, 17) int32
train_image_idxs <class 'numpy.ndarray'> (400135,) int32
val_captions <class 'numpy.ndarray'> (195954, 17) int32
val_image_idxs <class 'numpy.ndarray'> (195954,) int32
train_features <class 'numpy.ndarray'> (82783, 512) float32
val_features <class 'numpy.ndarray'> (40504, 512) float32
idx_to_word <class 'list'> 1004
word_to_idx <class 'dict'> 1004
train_urls <class 'numpy.ndarray'> (82783,) <U63
val_urls <class 'numpy.ndarray'> (40504,) <U63
```

load数据

In [6]:
```
# Sample a minibatch and show the images and captions
batch_size = 3

captions, features, urls = sample_coco_minibatch(data, batch_size=batch_size)
for i, (caption, url) in enumerate(zip(captions, urls)):
    plt.imshow(image_from_url(url))
    plt.axis('off')
    caption_str = decode_captions(caption, data['idx_to_word'])
    plt.title(caption_str)
    plt.show()
```

<START> a person in the air on a snowboard doing a trick <END>

<START> a sidewalk <UNK> in a city <UNK> of street lights pedestrians and <UNK> signs <END>

<START> a group of men standing around a kitchen <END>

放几张出来看一下
上面那行字是这幅图片的描述，每张图片的描述文字都是<start>开始<end>结尾
我们这个作业要做的工作是输入一幅图片，然后自动生成这幅图片的描述文字

接下来完成cs231n/rnn_layers.py里面的内容
Rnn_step_forward
在rnn中对于每一步前向传播，我们输入一个隐藏状态h_{t-1}（上一个神经网络的输出）和一个外部输入x_t*得到*下一个隐藏状态h_t，以及该时刻的y_t

$$h_{t-1} = f_w(h_{t-1},\ x_t) = \tanh（W_{hh}h_{t-1} + W_{xh}x_t + b）$$

```
next_h, cache = None, None
##############################################################################
# TODO: Implement a single forward step for the vanilla RNN. Store the next  #
# hidden state and any values you need for the backward pass in the next_h    #
# and cache variables respectively.                                          #
##############################################################################
temp1 = x.dot(Wx)
temp2 = prev_h.dot(Wh)
temp = temp1 + temp2 + b
next_h = np.tanh(temp)
cache = (x,prev_h,Wx,Wh,temp)
##############################################################################
#                                END OF YOUR CODE                            #
##############################################################################
return next_h, cache
```

对着公式写就行，cache保存下来留着反向传播用

In [7]:
```
N, D, H = 3, 10, 4

x = np.linspace(-0.4, 0.7, num=N*D).reshape(N, D)
prev_h = np.linspace(-0.2, 0.5, num=N*H).reshape(N, H)
Wx = np.linspace(-0.1, 0.9, num=D*H).reshape(D, H)
Wh = np.linspace(-0.3, 0.7, num=H*H).reshape(H, H)
b = np.linspace(-0.2, 0.4, num=H)

next_h, _ = rnn_step_forward(x, prev_h, Wx, Wh, b)
expected_next_h = np.asarray([
  [-0.58172089, -0.50182032, -0.41232771, -0.31410098],
  [ 0.66854692,  0.79562378,  0.87755553,  0.92795967],
  [ 0.97934501,  0.99144213,  0.99646691,  0.99854353]])

print('next_h error: ', rel_error(expected_next_h, next_h))
```

```
next_h error:  6.292421426471037e-09
```

验证step_forward的准确性

下面完成cs231n/rnn_layers.py里面的内容

Rnn_step_backward

假设最终的损失函数为E，并且令a=$W_{hh}h_{t-1}+W_{xh}x_t+b$，则有：

$$\frac{dE}{dh_{t-1}}=\frac{dE}{dh_t}\frac{dh_t}{dh_{t-1}}=\frac{dE}{dh_t}\frac{dh_t}{da}\frac{da}{dh_{t-1}}$$

```
dx, dprev_h, dWx, dWh, db = None, None, None, None, None
##############################################################################
# TODO: Implement the backward pass for a single step of a vanilla RNN.     #
#
# HINT: For the tanh function, you can compute the local derivative in terms #
# of the output value from tanh.
##############################################################################
x,prev_h,Wx,Wh,temp = cache
N,H = dnext_h.shape[0],dnext_h.shape[1]
temp_all = np.ones((N,H)) - np.square(np.tanh(temp))
delta = temp_all * dnext_h          #先对整个tanh求导
dx = delta.dot(Wx.T)
dWx = (x.T).dot(delta)
dprev_h = delta.dot(Wh.T)
dWh = (prev_h.T).dot(delta)
db = (np.sum(delta,axis=0)).T
##############################################################################
#                                                END OF YOUR CODE
##############################################################################
return dx, dprev_h, dWx, dWh, db
```

同样也是对着公式写

In [8]:
```
from cs231n.rnn_layers import rnn_step_forward, rnn_step_backward
np.random.seed(231)
N, D, H = 4, 5, 6
x = np.random.randn(N, D)
h = np.random.randn(N, H)
Wx = np.random.randn(D, H)
Wh = np.random.randn(H, H)
b = np.random.randn(H)

out, cache = rnn_step_forward(x, h, Wx, Wh, b)

dnext_h = np.random.randn(*out.shape)

fx = lambda x: rnn_step_forward(x, h, Wx, Wh, b)[0]
fh = lambda prev_h: rnn_step_forward(x, h, Wx, Wh, b)[0]
fWx = lambda Wx: rnn_step_forward(x, h, Wx, Wh, b)[0]
fWh = lambda Wh: rnn_step_forward(x, h, Wx, Wh, b)[0]
fb = lambda b: rnn_step_forward(x, h, Wx, Wh, b)[0]

dx_num = eval_numerical_gradient_array(fx, x, dnext_h)
dprev_h_num = eval_numerical_gradient_array(fh, h, dnext_h)
dWx_num = eval_numerical_gradient_array(fWx, Wx, dnext_h)
dWh_num = eval_numerical_gradient_array(fWh, Wh, dnext_h)
db_num = eval_numerical_gradient_array(fb, b, dnext_h)

dx, dprev_h, dWx, dWh, db = rnn_step_backward(dnext_h, cache)

print('dx error: ', rel_error(dx_num, dx))
print('dprev_h error: ', rel_error(dprev_h_num, dprev_h))
print('dWx error: ', rel_error(dWx_num, dWx))
print('dWh error: ', rel_error(dWh_num, dWh))
print('db error: ', rel_error(db_num, db))
```

```
dx error:  3.004984354606141e-10
dprev_h error:  2.633205333189269e-10
dWx error:  9.684083573724284e-10
dWh error:  3.355162782632426e-10
db error:  1.5956895526227225e-11
```

验证step_backward的准确性

下面完成cs231n/rnn_layers.py里面的内容

Rnn_forward

```
h, cache = None, None
##############################################################################
# TODO: Implement forward pass for a vanilla RNN running on a sequence of    #
# input data. You should use the rnn_step_forward function that you defined  #
# above. You can use a for loop to help compute the forward pass.            #
##############################################################################
N,T,D = x.shape
H = h0.shape[1]
prev_h = h0
h1 = np.empty([N,T,H])
h2 = np.empty([N,T,H])
h3 = np.empty([N,T,H])
for i in range(0,T):
      h,cache = rnn_step_forward(x[:,i,:], prev_h, Wx, Wh, b)
      h1[:,i,:] = prev_h
      prev_h = h
      h2[:,i,:] = h
      h3[:,i,:] = cache[4]
cache = (x,h1,Wx,Wh,h3)
##############################################################################
#                                                END OF YOUR CODE            #
##############################################################################
return h2, cache
```

x（N,T,D）N表示样本数，T表示时间步，D表示样本维度

H为隐藏状态维度

整个rnn的前向传播就是每一个时间步执行一次step_forward，并且把最后一次的cache保存下来

在每一步前向传播的过程中，Wx和Wh都是不变的，这里是rnn做的一个参数共享的假设

```
In [11]: N, T, D, H = 2, 3, 4, 5

         x = np.linspace(-0.1, 0.3, num=N*T*D).reshape(N, T, D)
         h0 = np.linspace(-0.3, 0.1, num=N*H).reshape(N, H)
         Wx = np.linspace(-0.2, 0.4, num=D*H).reshape(D, H)
         Wh = np.linspace(-0.4, 0.1, num=H*H).reshape(H, H)
         b = np.linspace(-0.7, 0.1, num=H)

         h, _ = rnn_forward(x, h0, Wx, Wh, b)
         expected_h = np.asarray([
           [
             [-0.42070749, -0.27279261, -0.11074945,  0.05740409,  0.22236251],
             [-0.39525808, -0.22554661, -0.0409454 ,  0.14649412,  0.32397316],
             [-0.42305111, -0.24223728, -0.04287027,  0.15997045,  0.35014525],
           ],
           [
             [-0.55857474, -0.39065825, -0.19198182,  0.02378408,  0.23735671],
             [-0.27150199, -0.07088804,  0.13562939,  0.33099728,  0.50158768],
             [-0.51014825, -0.30524429, -0.06755202,  0.17806392,  0.40333043]]])
         print('h error: ', rel_error(expected_h, h))

         h error:  7.728466180186066e-08
```

验证rnn_forward的准确性

下面完成cs231n/rnn_layers.py里面的内容

Rnn_backward

```
dx, dh0, dWx, dWh, db = None, None, None, None, None
##############################################################################
# TODO: Implement the backward pass for a vanilla RNN running an entire      #
# sequence of data. You should use the rnn_step_backward function that you   #
# defined above. You can use a for loop to help compute the backward pass.   #
##############################################################################
N,T,H = dh.shape
x = cache[0]
D = x.shape[2]
dWx = np.zeros((D,H))
dWh = np.zeros((H,H))
db = np.zeros(H)
dout = dh
dx = np.empty([N,T,D])
dh0 = np.empty([N,T,H])
dh_now = np.zeros((N,H))
for j in range(0,T):
      i = T-1-j
      dh_now = dh_now + dout[:,i,:]
      cache_t = (cache[0][:,i,:],cache[1][:,i,:],cache[2],cache[3],cache[4][:,i,:])
      dx_temp,dprev_h_temp,dWx_temp,dWh_temp,db_temp = rnn_step_backward(dh_now,cache_t)
      dh_now = dprev_h_temp
      dx[:,i,:] = dx_temp
      dWx += dWx_temp
      dWh += dWh_temp
      db += db_temp
      # 因为每一层都使用了共享参数Wx,Wh,b，所以这三个的梯度为所有层的和
dh0 = dh_now
##############################################################################
#                                                END OF YOUR CODE            #
##############################################################################
return dx, dh0, dWx, dWh, db
```

这里把刚刚前向传播的cache内容取出来有点麻烦，因为每一层都共享了Wx，Wh，b所以对每一层（时间步）求完梯度之后要求和，得到总的梯度。

In [17]:
```
np.random.seed(231)

N, D, T, H = 2, 3, 10, 5

x = np.random.randn(N, T, D)
h0 = np.random.randn(N, H)
Wx = np.random.randn(D, H)
Wh = np.random.randn(H, H)
b = np.random.randn(H)

out, cache = rnn_forward(x, h0, Wx, Wh, b)

dout = np.random.randn(*out.shape)

dx, dh0, dWx, dWh, db = rnn_backward(dout, cache)

fx = lambda x: rnn_forward(x, h0, Wx, Wh, b)[0]
fh0 = lambda h0: rnn_forward(x, h0, Wx, Wh, b)[0]
fWx = lambda Wx: rnn_forward(x, h0, Wx, Wh, b)[0]
fWh = lambda Wh: rnn_forward(x, h0, Wx, Wh, b)[0]
fb = lambda b: rnn_forward(x, h0, Wx, Wh, b)[0]

dx_num = eval_numerical_gradient_array(fx, x, dout)
dh0_num = eval_numerical_gradient_array(fh0, h0, dout)
dWx_num = eval_numerical_gradient_array(fWx, Wx, dout)
dWh_num = eval_numerical_gradient_array(fWh, Wh, dout)
db_num = eval_numerical_gradient_array(fb, b, dout)
```

```
print('dx error: ', rel_error(dx_num, dx))
print('dh0 error: ', rel_error(dh0_num, dh0))
print('dWx error: ', rel_error(dWx_num, dWx))
print('dWh error: ', rel_error(dWh_num, dWh))
print('db error: ', rel_error(db_num, db))
```

```
dx error:  1.9817764131204256e-09
dh0 error:  3.381247306333069e-09
dWx error:  7.2584869558584315e-09
dWh error:  1.2801162187950054e-07
db error:  4.36726574107421e-10
```

验证rnn_backward的准确性

下面完成word_embedding

每一个词对应一个向量，词和向量的对应关系也会进行学习

下面完成cs231n/rnn_layers.py里面的内容

Word_embedding_forward

```
out, cache = None, None
##############################################################################
# TODO: Implement the forward pass for word embeddings.
#
# HINT: This can be done in one line using NumPy's array indexing.
##############################################################################
out = W[x,:]
# 总共V个词，每个词用一个1*D向量表示
# x为N个样本，每个样本序列长度为T，序列中每个元素为1*D向量
# 该函数作用为，将输入的x进行编码。
cache = (W,x)
##############################################################################
#                               END OF YOUR CODE
##############################################################################
return out, cache
```

W（V,D）是词库，x（N,T）是样本，N为样本个数，T为每个样本句子的长度也就是词的个数，然后根据x中的每个值在词库中进行编码得到out（N,T,D）

In [19]:
```
N, T, V, D = 2, 4, 5, 3

x = np.asarray([[0, 3, 1, 2], [2, 1, 0, 3]])
W = np.linspace(0, 1, num=V*D).reshape(V, D)

out, _ = word_embedding_forward(x, W)
expected_out = np.asarray([
 [[ 0.,          0.07142857,  0.14285714],
  [ 0.64285714,  0.71428571,  0.78571429],
  [ 0.21428571,  0.28571429,  0.35714286],
  [ 0.42857143,  0.5,         0.57142857]],
 [[ 0.42857143,  0.5,         0.57142857],
  [ 0.21428571,  0.28571429,  0.35714286],
  [ 0.,          0.07142857,  0.14285714],
  [ 0.64285714,  0.71428571,  0.78571429]]])

print('out error: ', rel_error(expected_out, out))
```

```
(2, 4)
out error:  1.0000000094736443e-08
```

验证word_embedding_forward的正确性

```
dW = None
##############################################################################
# TODO: Implement the backward pass for word embeddings.
#
# Note that words can appear more than once in a sequence.
# HINT: Look up the function np.add.at
##############################################################################
W,x = cache
dW = np.zeros_like(W)
np.add.at(dW,x,dout)
##############################################################################
#                                                END OF YOUR CODE
##############################################################################
return dW
```

Word_embedding的反向传播

在x的位置上，加上前面传下来的梯度dout

In [20]:
```
np.random.seed(231)

N, T, V, D = 50, 3, 5, 6
x = np.random.randint(V, size=(N, T))
W = np.random.randn(V, D)

out, cache = word_embedding_forward(x, W)
dout = np.random.randn(*out.shape)
dW = word_embedding_backward(dout, cache)

f = lambda W: word_embedding_forward(x, W)[0]
dW_num = eval_numerical_gradient_array(f, W, dout)

print('dW error: ', rel_error(dW, dW_num))
```

```
dW error:  3.2774595693100364e-12
```

验证正确性

接下来是Temporal Affine layer，在每个时间步将隐藏状态转化为词库里对应词向量的分数，用于更新词向量，前向和后向都已经写好了。

In [21]:
```
np.random.seed(231)

# Gradient check for temporal affine layer
N, T, D, M = 2, 3, 4, 5
x = np.random.randn(N, T, D)
w = np.random.randn(D, M)
b = np.random.randn(M)

out, cache = temporal_affine_forward(x, w, b)

dout = np.random.randn(*out.shape)

fx = lambda x: temporal_affine_forward(x, w, b)[0]
fw = lambda w: temporal_affine_forward(x, w, b)[0]
fb = lambda b: temporal_affine_forward(x, w, b)[0]

dx_num = eval_numerical_gradient_array(fx, x, dout)
dw_num = eval_numerical_gradient_array(fw, w, dout)
db_num = eval_numerical_gradient_array(fb, b, dout)

dx, dw, db = temporal_affine_backward(dout, cache)

print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
dx error:  2.9215854231394017e-10
dw error:  1.5772169135951167e-10
db error:  3.252200556967514e-11
```

验证结果

In [22]:
```
# Sanity check for temporal softmax loss
from cs231n.rnn_layers import temporal_softmax_loss

N, T, V = 100, 1, 10

def check_loss(N, T, V, p):
    x = 0.001 * np.random.randn(N, T, V)
    y = np.random.randint(V, size=(N, T))
    mask = np.random.rand(N, T) <= p
    print(temporal_softmax_loss(x, y, mask)[0])

check_loss(100, 1, 10, 1.0)   # Should be about 2.3
check_loss(100, 10, 10, 1.0)  # Should be about 23
check_loss(5000, 10, 10, 0.1) # Should be about 2.3

# Gradient check for temporal softmax loss
N, T, V = 7, 8, 9

x = np.random.randn(N, T, V)
y = np.random.randint(V, size=(N, T))
mask = (np.random.rand(N, T) > 0.5)

loss, dx = temporal_softmax_loss(x, y, mask, verbose=False)

dx_num = eval_numerical_gradient(lambda x: temporal_softmax_loss(x, y, mask)[0], x, verbose=False)

print('dx error: ', rel_error(dx, dx_num))
```

```
2.3027781774290146
23.025985953127226
2.2643611790293394
dx error:  2.583585303524283e-08
```

这是验证softmax的正确性，这个函数已经给出来了

接下来完成cs231n/classifier/rnn.py

完成loss部分

```
################################################################################
# TODO: Implement the forward and backward passes for the CaptioningRNN.     #
# In the forward pass you will need to do the following:                       #
# (1) Use an affine transformation to compute the initial hidden state       #
#       from the image features. This should produce an array of shape (N, H)#
# (2) Use a word embedding layer to transform the words in captions_in       #
#       from indices to vectors, giving an array of shape (N, T, W).           #
# (3) Use either a vanilla RNN or LSTM (depending on self.cell_type) to      #
#       process the sequence of input word vectors and produce hidden state  #
#       vectors for all timesteps, producing an array of shape (N, T, H).      #
# (4) Use a (temporal) affine transformation to compute scores over the      #
#       vocabulary at every timestep using the hidden states, giving an        #
#       array of shape (N, T, V).
# (5) Use (temporal) softmax to compute loss using captions_out, ignoring  #
#       the points where the output word is <NULL> using the mask above.       #
#
# In the backward pass you will need to compute the gradient of the loss     #
# with respect to all model parameters. Use the loss and grads variables     #
# defined above to store loss and gradients; grads[k] should give the        #
# gradients for self.params[k].
#
# Note also that you are allowed to make use of functions from layers.py     #
# in your implementation, if needed.
################################################################################
N,D = features.shape
out,cache_affine = temporal_affine_forward(features.reshape(N,1,D),W_proj,b_proj)
N,t,H = out.shape
h0 = out.reshape(N,H)
# 图像特征转换成初始隐藏状态，

word_out,cache_word = word_embedding_forward(captions_in,W_embed)
# 将captions_in里面的单词转换成词向量
```

```
#hidden, cache_hidden = rnn_forward(word_out, h0, Wx, Wh, b)
# 算出RNN每一个隐藏层的状态
hidden, cache_hidden = lstm_forward(word_out, h0, Wx, Wh, b)
#LSTM

out_score, cache_score = temporal_affine_forward(hidden, W_vocab, b_vocab)
# 计算每一个隐藏层的得分

loss, dx = temporal_softmax_loss(out_score[:, :, :], captions_out, mask, verbose=False)
# 计算loss

dx_affine, dW_vocab, db_vocab = temporal_affine_backward(dx, cache_score)
grads['W_vocab'] = dW_vocab
grads['b_vocab'] = db_vocab

#dx_hidden, dh0, dWx, dWh, db = rnn_backward(dx_affine, cache_hidden)
#RNN
dx_hidden, dh0, dWx, dWh, db = lstm_backward(dx_affine, cache_hidden)
#LSTM

grads['Wx'] = dWx
grads['Wh'] = dWh
grads['b'] = db

dW_embed = word_embedding_backward(dx_hidden, cache_word)
grads['W_embed'] = dW_embed

_, dW_proj, db_proj = temporal_affine_backward(dh0.reshape(N, t, H), cache_affine)
grads['W_proj'] = dW_proj
grads['b_proj'] = db_proj
############################################################################
#                                               END OF YOUR CODE
############################################################################
```

这里图片特征是已经处理好给出来的，先让图片特征做一个仿射变换作为初始的隐藏状态

然后将标注转换成词向量

（这里代码有的是lstm的，这是下一章的内容，由于是rnn的进化版这两个作业有很大部分的重合，不过代码里面都做了注释就不在这里改过来了）

然后开始前向传播，算出每一层的隐藏状态

隐藏状态经过仿射变换得出每一个隐藏层的得分（这个得分是根据已有的标注来计算出来的）

用softmax计算loss

由于最后一层是仿射变换层，先对这一层求梯度，再逐层反向传播，再对词向量嵌入层做反向传播，最后对一开始的仿射变换层做反向传播。

```
############################################################################
# TODO: Implement test-time sampling for the model. You will need to       #
# initialize the hidden state of the RNN by applying the learned affine    #
# transform to the input image features. The first word that you feed to   #
# the RNN should be the <START> token; its value is stored in the          #
# variable self._start. At each timestep you will need to do to:           #
# (1) Embed the previous word using the learned word embeddings            #
# (2) Make an RNN step using the previous hidden state and the embedded    #
#       current word to get the next hidden state.                         #
# (3) Apply the learned affine transformation to the next hidden state to  #
#       get scores for all words in the vocabulary                         #
# (4) Select the word with the highest score as the next word, writing it  #
#       (the word index) to the appropriate slot in the captions variable  #
#                                                                          #
# For simplicity, you do not need to stop generating after an <END> token  #
# is sampled, but you can if you want to.                                  #
#                                                                          #
# HINT: You will not be able to use the rnn_forward or lstm_forward        #
# functions; you'll need to call rnn_step_forward or lstm_step_forward in  #
# a loop.                                                                  #
#                                                                          #
# NOTE: we are still working over minibatches in this function. Also if    #
# you are using an LSTM, initialize the first cell state to zeros.         #
############################################################################
N, D = features.shape
out, cache_affine = temporal_affine_forward(features.reshape(N, 1, D), W_proj, b_proj)
N, t, H = out.shape
h0 = out.reshape(N, H)
h = h0
```

```
x0 = W_embed[[1,1],:]
x_input = x0
# x_input为<START>
captions[:,0] = [1,1]
# 每个captions第一个字符都为<START>

prev_c = np.zeros_like(h)
for i in range(0,max_length - 1):
      #next_h,_ = rnn_step_forward(x_input,h,Wx,Wh,b)
      #RNN
      next_h,next_c,cache = lstm_step_forward(x_input,h,prev_c,Wx,Wh,b)
      prev_c = next_c


      out_score,cache_score = temporal_affine_forward(next_h.reshape(N,1,H),W_vocab,b_vocab)
      index = np.argmax(out_score,axis=2)
      x_input = np.squeeze(W_embed[index,:])
      h = next_h
      captions[:,i+1] = np.squeeze(index)
##############################################################################
#                                             END OF YOUR CODE
##############################################################################
return captions
```

61给图片的features生成对应的标注

首先定义x0在词库里面的位置，然后让每个标注的第一个词都指向这个位置，即第一个词永远是<start>

然后将x0做仿射变换生成初始隐藏状态，前向传播，再做仿射变换算出score然后找出得分最高的那个词作为该时间步的标注，然后将这个词作为下一步的输入，直到生成所有的词为止。

In [26]:
```
N, D, W, H = 10, 20, 30, 40
word_to_idx = {'<NULL>': 0, 'cat': 2, 'dog': 3}
V = len(word_to_idx)
T = 13

model = CaptioningRNN(word_to_idx,
          input_dim=D,
          wordvec_dim=W,
          hidden_dim=H,
          cell_type='rnn',
          dtype=np.float64)

# Set all model parameters to fixed values
for k, v in model.params.items():
    model.params[k] = np.linspace(-1.4, 1.3, num=v.size).reshape(*v.shape)

features = np.linspace(-1.5, 0.3, num=(N * D)).reshape(N, D)
captions = (np.arange(N * T) % V).reshape(N, T)

loss, grads = model.loss(features, captions)
expected_loss = 9.83235591003

print('loss: ', loss)
print('expected loss: ', expected_loss)
print('difference: ', abs(loss - expected_loss))
```

```
loss:  9.832355910027388
expected loss:  9.83235591003
difference:  2.611244553918368e-12
```

这是验证前面的loss

In [27]:
```
np.random.seed(231)

batch_size = 2
timesteps = 3
input_dim = 4
wordvec_dim = 5
hidden_dim = 6
word_to_idx = {'<NULL>': 0, 'cat': 2, 'dog': 3}
vocab_size = len(word_to_idx)

captions = np.random.randint(vocab_size, size=(batch_size, timesteps))
features = np.random.randn(batch_size, input_dim)

model = CaptioningRNN(word_to_idx,
          input_dim=input_dim,
          wordvec_dim=wordvec_dim,
          hidden_dim=hidden_dim,
          cell_type='rnn',
          dtype=np.float64,
        )

loss, grads = model.loss(features, captions)

for param_name in sorted(grads):
    f = lambda _: model.loss(features, captions)[0]
    param_grad_num = eval_numerical_gradient(f, model.params[param_name], verbose=False, h=1e-6)
    e = rel_error(param_grad_num, grads[param_name])
    print('%s relative error: %e' % (param_name, e))
```

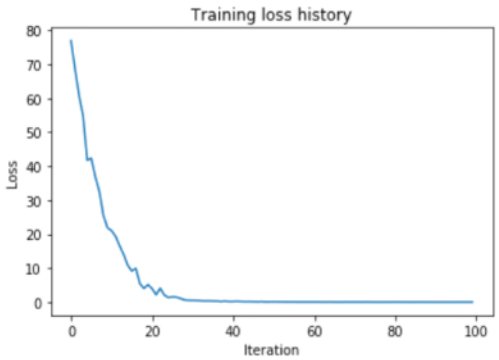
```
W_embed relative error: 2.331072e-09
W_proj relative error: 9.974424e-09
W_vocab relative error: 4.274378e-09
Wh relative error: 5.954804e-09
Wx relative error: 8.455229e-07
b relative error: 8.001353e-10
b_proj relative error: 6.260036e-09
b_vocab relative error: 6.918525e-11
```

验证这个captioningrnn的正确性

In [28]:
```
np.random.seed(231)

small_data = load_coco_data(max_train=50)

small_rnn_model = CaptioningRNN(
          cell_type='rnn',
          word_to_idx=data['word_to_idx'],
          input_dim=data['train_features'].shape[1],
          hidden_dim=512,
          wordvec_dim=256,
        )

small_rnn_solver = CaptioningSolver(small_rnn_model, small_data,
           update_rule='adam',
           num_epochs=50,
           batch_size=25,
           optim_config={
             'learning_rate': 5e-3,
           },
           lr_decay=0.95,
           verbose=True, print_every=10,
         )

small_rnn_solver.train()
```

```
# Plot the training losses
plt.plot(small_rnn_solver.loss_history)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Training loss history')
plt.show()
```

```
(Iteration 1 / 100) loss: 76.913487
(Iteration 11 / 100) loss: 21.063181
(Iteration 21 / 100) loss: 4.016191
(Iteration 31 / 100) loss: 0.567074
(Iteration 41 / 100) loss: 0.239429
(Iteration 51 / 100) loss: 0.162023
(Iteration 61 / 100) loss: 0.111542
(Iteration 71 / 100) loss: 0.097583
(Iteration 81 / 100) loss: 0.099098
(Iteration 91 / 100) loss: 0.073980
```



用他已经写好的solver测试模型

In [31]:
```
for split in ['train', 'val']:
    minibatch = sample_coco_minibatch(small_data, split=split, batch_size=2)
    gt_captions, features, urls = minibatch
    gt_captions = decode_captions(gt_captions, data['idx_to_word'])

    sample_captions = small_rnn_model.sample(features)
    sample_captions = decode_captions(sample_captions, data['idx_to_word'])

    for gt_caption, sample_caption, url in zip(gt_captions, sample_captions, urls):
        plt.imshow(image_from_url(url))
        plt.title('%s\n%s\nGT:%s' % (split, sample_caption, gt_caption))
        plt.axis('off')
        plt.show()
```

train
<START> kids and a man walking down the sidewalk with suitcases <END>
GT:<START> kids and a man walking down the sidewalk with suitcases <END>

train
<START> there is a male surfer coming out of the water <END>
GT:<START> there is a male surfer coming out of the water <END>

val
<START> various <UNK> with <UNK> <END>
GT:<START> an elephant and baby elephant walk towards the water <END>

val
<START> man truck woman on the <UNK> in a <UNK> <END>
GT:<START> a <UNK> <UNK> through a <UNK> <UNK> with benches <END>

看一下最终的效果，其实并不是很理想啊

LSTM

2019年5月1日 19:57

LSTM原理现在前面说好了，这一部分解释是我在做数模比赛时敲的，参考csdn一篇博客
https://blog.csdn.net/zhangbaoanhadoop/article/details/81952284

循环神经网络（Recurrent Neural Network）是一种节点定向连接成环的人工神经网络。这种网络的内部状态可以展示动态时序行为。不同于传统神经网络的是，RNN可以利用它内部的记忆来处理任意时序的输入序列，这让它可以更容易处理时间序列预测问题。但是普通的RNN存在一定的弊端，由于原始RNN模型也是使用BP算法进行参数的训练，梯度消失和梯度爆炸的问题不仅得不到解决，反而在内部的迭代过程中得到了放大。

LSTM的出现解决了上述所说的问题。LSTM的结构与RNN相似，都为链式结构。不同的是，RNN内部的每个记忆单元使用的是单一网络层（见图4-1），而LSTM对RNN的每个记忆单元都增加了四个门控结构见图（4-2）。[4][5][7]

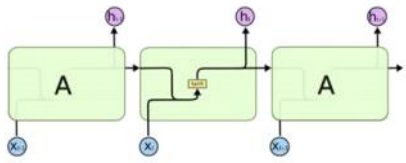


图 4-1 RNN结构图

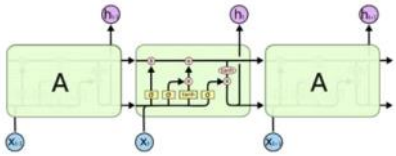


图 4-2 LSTM结构图

从图4-2中可以看出，在每个序列索引位置t时刻向前传播的除了和RNN一样的隐藏状态h_t，还多了另一个隐藏状态（以下记为细胞状态C_t），如图4-2中的长横线。如图4-3所示：

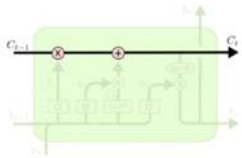


图 4-3 LSTM细胞状态

除了细胞状态之外，LSTM还多了三个门控结构，我们称之为遗忘门，输入门和输出门。

遗忘门（forget gate）是控制记忆单元是否遗忘上一层的隐藏细胞状态。遗忘门子结构如图4-4所示：

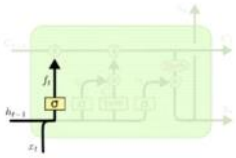


图 4-4 LSTM遗忘门

遗忘门输入之前的隐藏状态h_{t-1}，和当前序列数据x_t，通过激活函数sigmoid输出上一层细胞状态的遗忘概率f_t。

输入门（input gate）负责处理当前序列位置的输入，它的结构如图4-5：

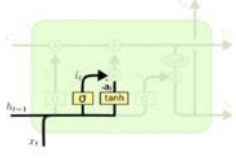


图 4-5 LSTM输入门

输入门输入之前的隐藏状态h_{t-1}，和当前序列数据x_t，分别通过激活函数sigmoid和tanh得到i_t和a_t，再相乘，得到结果用于更新细胞状态。

状态更新结构如图4-6：

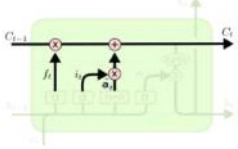


图 4-6 LSTM状态更新

输入之前的细胞状态c_{t-1}和遗忘门输出结果f_t以及输入门的输出结果i_t和a_t来更新细胞状态。

输出门结构如图4-7：

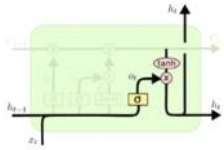


图 4-7 LSTM输出门

输出门用来更新隐藏状态，分为两个部分，第一部分输入之前的隐藏状态h_{t-1}和当前序列输入x_t通过激活函数sigmoid得到o_t，第二部分输入状态更新的输出c_t通过激活函数tanh得到tanh(c_t)，两部分结果做Hadmard积[6]得到当前隐藏状态h_t。

最后更新当前序列引索预测输出。

LSTM的前向传播算法总结为：[7]

·更新遗忘门输出：

$f^{(t)} = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f)$

·更新输入门两部分输出：

$i^{(t)} = \sigma(W_i + U_i x^{(t)} + b_i)$

$a^{(t)} = \tanh(W_a h^{(t-1)} + U_a x^{(t)} + b_a)$

·更新细胞状态 ：

$C^{(t)} = C^{(t-1)} \odot f^{(t)} + i^{(t)} \odot a^{(t)}$

·更新输出门输出：

$o^{(t)} = \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o)$

$h^{(t)} = o^{(t)} \odot \tanh(C^{(t)})$

·更新当前序列索引预测输出：

$y^{(t)} = \sigma(V h^{(t)} + c)$

LSTM的后向算法通过链式法则可对参数梯度一一求解。

其实这部分作业和RNN一样，无非就是换了公式，后面训练的代码几乎都是一样的

然后来看作业部分

In [3]:
```
# As usual, a bit of setup
import time, os, json
import numpy as np
import matplotlib.pyplot as plt

from cs231n.gradient_check import eval_numerical_gradient, eval_numerical_gradient_array
from cs231n.rnn_layers import *
from cs231n.captioning_solver import CaptioningSolver
from cs231n.classifiers.rnn import CaptioningRNN
from cs231n.coco_utils import load_coco_data, sample_coco_minibatch, decode_captions
from cs231n.image_utils import image_from_url

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
    """ returns relative error """
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

In [4]:
```
# Load COCO data from disk; this returns a dictionary
# We'll work with dimensionality-reduced features for this notebook, but feel
# free to experiment with the original features by changing the flag below.
data = load_coco_data(pca_features=True)

# Print out all the keys and values from the data dictionary
for k, v in data.items():
    if type(v) == np.ndarray:
        print(k, type(v), v.shape, v.dtype)
    else:
        print(k, type(v), len(v))
```

```
train_captions <class 'numpy.ndarray'> (400135, 17) int32
train_image_idxs <class 'numpy.ndarray'> (400135,) int32
val_captions <class 'numpy.ndarray'> (195954, 17) int32
val_image_idxs <class 'numpy.ndarray'> (195954,) int32
train_features <class 'numpy.ndarray'> (82783, 512) float32
val_features <class 'numpy.ndarray'> (40504, 512) float32
idx_to_word <class 'list'> 1004
word_to_idx <class 'dict'> 1004
train_urls <class 'numpy.ndarray'> (82783,) <U63
val_urls <class 'numpy.ndarray'> (40504,) <U63
```

下面完成cs231n/rnn_layers/lstm_step_forward

```
next_h, next_c, cache = None, None, None
#############################################################################
# TODO: Implement the forward pass for a single timestep of an LSTM.          #
# You may want to use the numerically stable sigmoid implementation above.  #
#############################################################################
N,H = prev_h.shape
A = x.dot(Wx) + prev_h.dot(Wh) + b
ai = A[:,:H]
af = A[:,H:2*H]
ao = A[:,2*H:3*H]
ag = A[:,3*H:4*H]

i = sigmoid(ai)
f = sigmoid(af)
o = sigmoid(ao)
g = np.tanh(ag)

next_c =np.multiply(f,prev_c) + np.multiply(i,g)
next_h = np.multiply(o,np.tanh(next_c))

cache = (x,prev_h,prev_c,i,f,o,g,Wx,Wh,next_c,A)
#############################################################################
#                                                END OF YOUR CODE
#############################################################################

return next_h, next_c, cache
```

就把四个门的结果算出来就行，和rnn一样保存cache

```
In [5]: N, D, H = 3, 4, 5
        x = np.linspace(-0.4, 1.2, num=N*D).reshape(N, D)
        prev_h = np.linspace(-0.3, 0.7, num=N*H).reshape(N, H)
        prev_c = np.linspace(-0.4, 0.9, num=N*H).reshape(N, H)
        Wx = np.linspace(-2.1, 1.3, num=4*D*H).reshape(D, 4 * H)
        Wh = np.linspace(-0.7, 2.2, num=4*H*H).reshape(H, 4 * H)
        b = np.linspace(0.3, 0.7, num=4*H)

        next_h, next_c, cache = lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)

        expected_next_h = np.asarray([
            [ 0.24635157,  0.28610883,  0.32240467,  0.35525807,  0.38474904],
            [ 0.49223563,  0.55611431,  0.61507696,  0.66844003,  0.7159181 ],
            [ 0.56735664,  0.66310127,  0.74419266,  0.80889665,  0.858299  ]])
        expected_next_c = np.asarray([
            [ 0.32986176,  0.39145139,  0.451556,    0.51014116,  0.56717407],
            [ 0.66382255,  0.76674007,  0.87195994,  0.97902709,  1.08751345],
            [ 0.74192008,  0.90592151,  1.07717006,  1.25120233,  1.42395676]])

        print('next_h error: ', rel_error(expected_next_h, next_h))
        print('next_c error: ', rel_error(expected_next_c, next_c))

        next_h error:  5.7054131185818695e-09
        next_c error:  5.8143123088804145e-09
```

验证结果

```
dx, dprev_h, dprev_c, dWx, dWh, db = None, None, None, None, None, None
#############################################################################
# TODO: Implement the backward pass for a single timestep of an LSTM.         #
#                                                                           #
# HINT: For sigmoid and tanh you can compute local derivatives in terms of  #
# the output value from the nonlinearity.                                   #
#############################################################################
N,H = dnext_h.shape
prev_x,prev_h,prev_c,i,f,o,g,Wx,Wh,next_c,A = cache
ai = A[:,:H]
af = A[:,H:2*H]
ao = A[:,2*H:3*H]
ag = A[:,3*H:4*H]

# 到ct-1的梯度
dc_2 = np.multiply(dnext_c,f)
temp = np.multiply(dnext_h,o)
temp1 = np.ones_like(next_c) - np.square(np.tanh(next_c))
temp2 = np.multiply(temp1,f)
dc_1 = np.multiply(temp,temp2)
dprev_c = dc_1 + dc_2
#dE/dh * dh/dc
dE_dc = np.multiply(temp,temp1) + dnext_c

#计算di、df、do、dg
di = np.multiply(dE_dc,g)
dg = np.multiply(dE_dc,i)
df = np.multiply(dE_dc,prev_c)
do = np.multiply(dnext_h,np.tanh(next_c))
```

```
#到ai,af,ao,ag
dao = np.multiply(do,np.multiply(o,(np.ones_like(o) - o)))
daf = np.multiply(df,np.multiply(f,(np.ones_like(f) - f)))
dai = np.multiply(di,np.multiply(i,(np.ones_like(i) - i)))
dtanhg = np.ones_like(ag) - np.square(np.tanh(ag))
dag = np.multiply(dg,dtanhg)

#计算各参数梯度
dall = np.concatenate((dai,daf,dao,dag),axis=1)
dx = dall.dot(Wx.T)
dprev_h = dall.dot(Wh.T)
dWx = prev_x.T.dot(dall)
dWh = prev_h.T.dot(dall)
db = np.sum(dall,axis=0).T
################################################################################
#                                END OF YOUR CODE
################################################################################

return dx, dprev_h, dprev_c, dWx, dWh, db
```

后向传播

后向传播由于公式有点多，计算较为麻烦一些，在csdn上一篇博客

（https://blog.csdn.net/BigDataDigest/article/details/79314717）找了一下推导过程

虽然同样是利用了求导的链式法则，但是由于多了一条关于c_t传播的路径，LSTM的反向传播会比一般的递归神经网络复杂一些。假设传到当前单元的导数为$\frac{dE}{dh_t}$，$\frac{dE}{dc_t}$，我们先讨论如何计算$\frac{dE}{dc_{t-1}}$。$\frac{dE}{dc_{t-1}}$可以由两条路径产生，一条是通过$\frac{dE}{dc_t}$,一条是通过$\frac{dE}{dh_t}$.

$$\begin{aligned}\frac{dE}{dc_{t-1}} &= \frac{dE}{dh_t}\frac{dh_t}{dc_{t-1}} + \frac{dE}{dc_t}\frac{dc_t}{dc_{t-1}} \\ &= \frac{dE}{dh_t}\frac{dh_t}{dc_t}\frac{dc_t}{dc_{t-1}} + \frac{dE}{dc_t}\frac{dc_t}{dc_{t-1}}\end{aligned}$$

这里$\frac{dh_t}{dc_t} = o\frac{d\tanh c_t}{dc_t}$，$\frac{dc_t}{dc_{t-1}} = f$。这里的所有乘法都是逐项（element wise）相乘。为了计算关于$\frac{dE}{dh_{t-1}}$等梯度，我们先要得到到i, f, g, o这些节点的梯度。到o的梯度易得:

$$\frac{dE}{do} = \frac{dE}{dh_t}\frac{dh}{do}$$

其中$\frac{dh}{do} = \tanh c_t$。

到f的梯度需要注意的是依然有两条路径，

$$\begin{aligned}\frac{dE}{df} &= \frac{dE}{dh_t}\frac{dh_t}{df} + \frac{dc_t}{df} \\ &= \frac{dE}{dh_t}\frac{dh_t}{dc_t}\frac{dc_t}{df} + \frac{dc_t}{df}\end{aligned}$$

其中$\frac{dc_t}{df} = c_{t-1}$。

到i和g的梯度计算过程相似，下面以计算$\frac{dE}{di}$为例:

$$\begin{aligned}\frac{dE}{di} &= \frac{dE}{dh_t}\frac{dh_t}{di} + \frac{dc_t}{di} \\ &= \frac{dE}{dh_t}\frac{dh_t}{dc_t}\frac{dc_t}{di} + \frac{dc_t}{di}\end{aligned}$$

其中$\frac{dc_t}{di} = g$。

而o, g, i, f是a_o, a_g, a_i, a_g通告激活函数而得，所以我们有:

$$\frac{dE}{da_o} = \frac{dE}{do}\frac{do}{da_o} \qquad \frac{dE}{da_g} = \frac{dE}{dg}\frac{dg}{da_g}$$

$$\frac{dE}{da_i} = \frac{dE}{di}\frac{di}{da_i} \qquad \frac{dE}{da_f} = \frac{dE}{df}\frac{df}{da_f}$$

将$\frac{dE}{da_i}, \frac{dE}{da_i}, \frac{dE}{da_i}, \frac{dE}{da_i}$相连，我们得到$\frac{dE}{da}$。至此，之后求$\frac{dE}{dh_{t-1}}$，$\frac{dE}{dW_{hh}}$等梯度的步骤和之前求一般递归神经网络反向传播公式时相同，这里略去。注意以上的乘法都为逐项相乘。

In [6]:
```
np.random.seed(231)

N, D, H = 4, 5, 6
x = np.random.randn(N, D)
prev_h = np.random.randn(N, H)
prev_c = np.random.randn(N, H)
Wx = np.random.randn(D, 4 * H)
Wh = np.random.randn(H, 4 * H)
b = np.random.randn(4 * H)

next_h, next_c, cache = lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)

dnext_h = np.random.randn(*next_h.shape)
dnext_c = np.random.randn(*next_c.shape)

fx_h = lambda x: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[0]
fh_h = lambda h: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[0]
fc_h = lambda c: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[0]
fWx_h = lambda Wx: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[0]
fWh_h = lambda Wh: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[0]
fb_h = lambda b: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[0]

fx_c = lambda x: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[1]
fh_c = lambda h: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[1]
fc_c = lambda c: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[1]
fWx_c = lambda Wx: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[1]
fWh_c = lambda Wh: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[1]
fb_c = lambda b: lstm_step_forward(x, prev_h, prev_c, Wx, Wh, b)[1]
```

```
num_grad = eval_numerical_gradient_array

dx_num = num_grad(fx_h, x, dnext_h) + num_grad(fx_c, x, dnext_c)
dh_num = num_grad(fh_h, prev_h, dnext_h) + num_grad(fh_c, prev_h, dnext_c)
dc_num = num_grad(fc_h, prev_c, dnext_h) + num_grad(fc_c, prev_c, dnext_c)
dWx_num = num_grad(fWx_h, Wx, dnext_h) + num_grad(fWx_c, Wx, dnext_c)
dWh_num = num_grad(fWh_h, Wh, dnext_h) + num_grad(fWh_c, Wh, dnext_c)
db_num = num_grad(fb_h, b, dnext_h) + num_grad(fb_c, b, dnext_c)

dx, dh, dc, dWx, dWh, db = lstm_step_backward(dnext_h, dnext_c, cache)

print('dx error: ', rel_error(dx_num, dx))
print('dh error: ', rel_error(dh_num, dh))
print('dc error: ', rel_error(dc_num, dc))
print('dWx error: ', rel_error(dWx_num, dWx))
print('dWh error: ', rel_error(dWh_num, dWh))
print('db error: ', rel_error(db_num, db))
```

```
dx error:  6.141307149471403e-10
dh error:  3.3953235055372503e-10
dc error:  1.5221747946070454e-10
dWx error:  1.6933643922734908e-09
dWh error:  2.5561308517943814e-08
db error:  1.7349247160222088e-10
```

验证结果

```
h, cache = None, None
#############################################################################
# TODO: Implement the forward pass for an LSTM over an entire timeseries.    #
# You should use the lstm_step_forward function that you just defined.       #
#############################################################################
N,T,D = x.shape
H = h0.shape[1]
prev_h = h0

h = np.empty([N,T,H])
h2 = np.empty([N,T,H])
h3 = np.empty([N,T,H])
h4 = np.empty([N,T,H])
I = np.empty([N,T,H])
f = np.empty([N,T,H])
o = np.empty([N,T,H])
g = np.empty([N,T,H])
nc = np.empty([N,T,H])
A = np.empty([N,T,4*H])
prev_c = np.zeros_like(prev_h)
```

```
for i in range(0,T):
        h3[:,i,:] = prev_h
        h4[:,i,:] = prev_c
        next_h,next_c,cache_temp = lstm_step_forward(x[:,i,:],prev_h,prev_c,Wx,Wh,b)
        prev_h = next_h
        prev_c = next_c
        h2[:,i,:] = prev_h
        I[:,i,:] = cache_temp[3]
        # cache = (x,prev_h,prev_c,i,f,o,g,Wx,Wh,next_c,A)
        f[:,i,:] = cache_temp[4]
        o[:,i,:] = cache_temp[5]
        g[:,i,:] = cache_temp[6]
          print(cache_temp[9].shape)
        nc[:,i,:] = cache_temp[9]
          print(cache_temp[10].shape)
        A[:,i,:] = cache_temp[10]

h = h2
cache = (x,h3,h4,I,f,o,g,Wx,Wh,nc,A)
##############################################################################
#                                                END OF YOUR CODE
##############################################################################

return h, cache
```

In [7]:
```
N, D, H, T = 2, 5, 4, 3
x = np.linspace(-0.4, 0.6, num=N*T*D).reshape(N, T, D)
h0 = np.linspace(-0.4, 0.8, num=N*H).reshape(N, H)
Wx = np.linspace(-0.2, 0.9, num=4*D*H).reshape(D, 4 * H)
Wh = np.linspace(-0.3, 0.6, num=4*H*H).reshape(H, 4 * H)
b = np.linspace(0.2, 0.7, num=4*H)

h, cache = lstm_forward(x, h0, Wx, Wh, b)

expected_h = np.asarray([
 [[ 0.01764008,  0.01823233,  0.01882671,  0.0194232 ],
  [ 0.11287491,  0.12146228,  0.13018446,  0.13902939],
  [ 0.31358768,  0.33338627,  0.35304453,  0.37250975]],
 [[ 0.45767879,  0.4761092,   0.4936887,   0.51041945],
  [ 0.6704845,   0.69350089,  0.71486014,  0.7346449 ],
  [ 0.81733511,  0.83677871,  0.85403753,  0.86935314]]])
print('h error: ', rel_error(expected_h, h))
```

```
h error:  8.610537452106624e-08
```

验证结果

In [8]:
```
from cs231n.rnn_layers import lstm_forward, lstm_backward
np.random.seed(231)

N, D, T, H = 2, 3, 10, 6

x = np.random.randn(N, T, D)
h0 = np.random.randn(N, H)
Wx = np.random.randn(D, 4 * H)
Wh = np.random.randn(H, 4 * H)
b = np.random.randn(4 * H)

out, cache = lstm_forward(x, h0, Wx, Wh, b)

dout = np.random.randn(*out.shape)

dx, dh0, dWx, dWh, db = lstm_backward(dout, cache)

fx = lambda x: lstm_forward(x, h0, Wx, Wh, b)[0]
fh0 = lambda h0: lstm_forward(x, h0, Wx, Wh, b)[0]
fWx = lambda Wx: lstm_forward(x, h0, Wx, Wh, b)[0]
fWh = lambda Wh: lstm_forward(x, h0, Wx, Wh, b)[0]
fb = lambda b: lstm_forward(x, h0, Wx, Wh, b)[0]
```

```
dx_num = eval_numerical_gradient_array(fx, x, dout)
dh0_num = eval_numerical_gradient_array(fh0, h0, dout)
dWx_num = eval_numerical_gradient_array(fWx, Wx, dout)
dWh_num = eval_numerical_gradient_array(fWh, Wh, dout)
db_num = eval_numerical_gradient_array(fb, b, dout)

print('dx error: ', rel_error(dx_num, dx))
print('dh0 error: ', rel_error(dh0_num, dh0))
print('dWx error: ', rel_error(dWx_num, dWx))
print('dWh error: ', rel_error(dWh_num, dWh))
print('db error: ', rel_error(db_num, db))
```

```
dx error:  4.825038391506627e-09
dh0 error:  7.500933278661376e-09
dWx error:  1.751995067725175e-09
dWh error:  1.0853769933605458e-06
db error:  7.42754365004995e-10
```

验证结果

```
############################################################################
# TODO: Implement the forward and backward passes for the CaptioningRNN.     #
# In the forward pass you will need to do the following:                     #
# (1) Use an affine transformation to compute the initial hidden state       #
#       from the image features. This should produce an array of shape (N, H)#
# (2) Use a word embedding layer to transform the words in captions_in       #
#       from indices to vectors, giving an array of shape (N, T, W).            #
# (3) Use either a vanilla RNN or LSTM (depending on self.cell_type) to      #
#       process the sequence of input word vectors and produce hidden state  #
#       vectors for all timesteps, producing an array of shape (N, T, H).      #
# (4) Use a (temporal) affine transformation to compute scores over the      #
#       vocabulary at every timestep using the hidden states, giving an        #
#       array of shape (N, T, V).                                            #
# (5) Use (temporal) softmax to compute loss using captions_out, ignoring  #
#       the points where the output word is <NULL> using the mask above.       #
#                                                                            #
# In the backward pass you will need to compute the gradient of the loss     #
# with respect to all model parameters. Use the loss and grads variables     #
# defined above to store loss and gradients; grads[k] should give the        #
# gradients for self.params[k].                                              #
#                                                                            #
# Note also that you are allowed to make use of functions from layers.py     #
# in your implementation, if needed.                                         #
############################################################################
N,D = features.shape
out,cache_affine = temporal_affine_forward(features.reshape(N,1,D),W_proj,b_proj)
N,t,H = out.shape
h0 = out.reshape(N,H)
# 图像特征转换成初始隐藏状态，

word_out,cache_word = word_embedding_forward(captions_in,W_embed)
# 将captions_in里面的单词转换成词向量

#hidden,cache_hidden = rnn_forward(word_out,h0,Wx,Wh,b)
# 算出RNN每一个隐藏层的状态
hidden,cache_hidden = lstm_forward(word_out,h0,Wx,Wh,b)
#LSTM

out_score,cache_score = temporal_affine_forward(hidden,W_vocab,b_vocab)
# 计算每一个隐藏层的得分

loss,dx = temporal_softmax_loss(out_score[:,:,:],captions_out,mask,verbose=False)
# 计算loss

dx_affine,dW_vocab,db_vocab = temporal_affine_backward(dx,cache_score)
grads['W_vocab'] = dW_vocab
grads['b_vocab'] = db_vocab

#dx_hidden,dh0,dWx,dWh,db = rnn_backward(dx_affine,cache_hidden)
#RNN
dx_hidden,dh0,dWx,dWh,db = lstm_backward(dx_affine,cache_hidden)
#LSTM

grads['Wx'] = dWx
grads['Wh'] = dWh
grads['b'] = db

dW_embed = word_embedding_backward(dx_hidden,cache_word)
grads['W_embed'] = dW_embed

_,dW_proj,db_proj = temporal_affine_backward(dh0.reshape(N,t,H),cache_affine)
grads['W_proj'] = dW_proj
grads['b_proj'] = db_proj
############################################################################
#                                       END OF YOUR CODE                   #
############################################################################
```

```
##############################################################################
# TODO: Implement test-time sampling for the model. You will need to        #
# initialize the hidden state of the RNN by applying the learned affine     #
# transform to the input image features. The first word that you feed to    #
# the RNN should be the <START> token; its value is stored in the           #
# variable self._start. At each timestep you will need to do to:           #
# (1) Embed the previous word using the learned word embeddings             #
# (2) Make an RNN step using the previous hidden state and the embedded     #
#       current word to get the next hidden state.                          #
# (3) Apply the learned affine transformation to the next hidden state to   #
#       get scores for all words in the vocabulary                          #
# (4) Select the word with the highest score as the next word, writing it   #
#       (the word index) to the appropriate slot in the captions variable   #
#                                                                           #
# For simplicity, you do not need to stop generating after an <END> token   #
# is sampled, but you can if you want to.                                   #
#                                                                           #
# HINT: You will not be able to use the rnn_forward or lstm_forward         #
# functions; you'll need to call rnn_step_forward or lstm_step_forward in   #
# a loop.                                                                   #
#                                                                           #
# NOTE: we are still working over minibatches in this function. Also if     #
# you are using an LSTM, initialize the first cell state to zeros.          #
##############################################################################
N,D = features.shape
out,cache_affine = temporal_affine_forward(features.reshape(N,1,D),W_proj,b_proj)
N,t,H = out.shape
h0 = out.reshape(N,H)
h = h0
x0 = W_embed[[1,1],:]
x_input = x0
# x_input为<START>
captions[:,0] = [1,1]
# 每个captions第一个字符都为<START>

prev_c = np.zeros_like(h)
for i in range(0,max_length - 1):
      #next_h,_ = rnn_step_forward(x_input,h,Wx,Wh,b)
      #RNN
      next_h,next_c,cache = lstm_step_forward(x_input,h,prev_c,Wx,Wh,b)
      prev_c = next_c


      out_score,cache_score = temporal_affine_forward(next_h.reshape(N,1,H),W_vocab,b_vocab)
      index = np.argmax(out_score,axis=2)
      x_input = np.squeeze(W_embed[index,:])
      h = next_h
      captions[:,i+1] = np.squeeze(index)
##############################################################################
#                                             END OF YOUR CODE               #
##############################################################################
return captions
```

这部分和rnn的一样

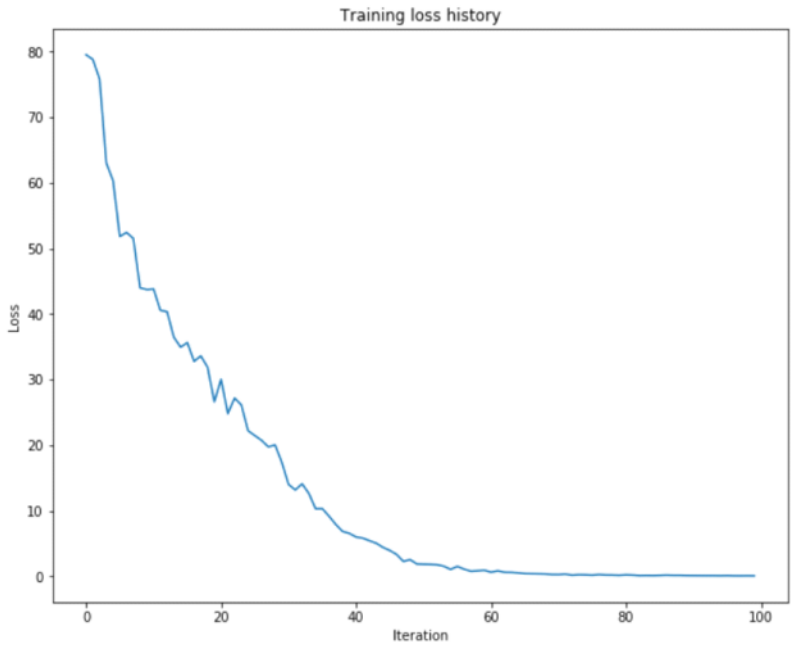
```
In [9]: N, D, W, H = 10, 20, 30, 40
        word_to_idx = {'<NULL>': 0, 'cat': 2, 'dog': 3}
        V = len(word_to_idx)
        T = 13

        model = CaptioningRNN(word_to_idx,
                  input_dim=D,
                  wordvec_dim=W,
                  hidden_dim=H,
                  cell_type='lstm',
                  dtype=np.float64)

        # Set all model parameters to fixed values
        for k, v in model.params.items():
          model.params[k] = np.linspace(-1.4, 1.3, num=v.size).reshape(*v.shape)

        features = np.linspace(-0.5, 1.7, num=N*D).reshape(N, D)
        captions = (np.arange(N * T) % V).reshape(N, T)

        loss, grads = model.loss(features, captions)
        expected_loss = 9.82445935443

        print('loss: ', loss)
        print('expected loss: ', expected_loss)
        print('difference: ', abs(loss - expected_loss))

        loss:  9.82445935443226
        expected loss:  9.82445935443
        difference:  2.261302256556519e-12
```

验证结果

In [10]:
```
np.random.seed(231)

small_data = load_coco_data(max_train=50)

small_lstm_model = CaptioningRNN(
          cell_type='lstm',
          word_to_idx=data['word_to_idx'],
          input_dim=data['train_features'].shape[1],
          hidden_dim=512,
          wordvec_dim=256,
          dtype=np.float32,
        )

small_lstm_solver = CaptioningSolver(small_lstm_model, small_data,
           update_rule='adam',
           num_epochs=50,
           batch_size=25,
           optim_config={
             'learning_rate': 5e-3,
           },
           lr_decay=0.995,
           verbose=True, print_every=10,
         )

small_lstm_solver.train()
```

```
# Plot the training losses
plt.plot(small_lstm_solver.loss_history)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Training loss history')
plt.show()
```

```
(25, 512)
(25, 17)
***
(Iteration 1 / 100) loss: 79.551150
(25, 512)
(25, 17)
***
(25, 512)
(25, 17)
***
(25, 512)
(25, 17)
***
(25, 512)
(25, 17)
***
(25, 512)
(25, 17)
***
(25, 512)
```



In [20]:
```
for split in ['train', 'val']:
    minibatch = sample_coco_minibatch(small_data, split=split, batch_size=2)
    gt_captions, features, urls = minibatch
    gt_captions = decode_captions(gt_captions, data['idx_to_word'])

    sample_captions = small_lstm_model.sample(features)
    sample_captions = decode_captions(sample_captions, data['idx_to_word'])

    for gt_caption, sample_caption, url in zip(gt_captions, sample_captions, urls):
        plt.imshow(image_from_url(url))
        plt.title('%s\n%s\nGT:%s' % (split, sample_caption, gt_caption))
        plt.axis('off')
        plt.show()
```

train
<START> many people standing near boxes of many apples <END>
GT:<START> many people standing near boxes of many apples <END>

train
<START> a surfer rides a large wave while the sun <UNK> the <UNK> <END>
GT:<START> a surfer rides a large wave while the sun <UNK> the <UNK> <END>

val
<START> tile five grazing grazing grazing sleeping cute cute dog standing on a the ground near a busy <END>
GT:<START> a bowl of chicken and vegetables is shown <END>

val
<START> an open refrigerator people standing with a man on the <UNK> <END>
GT:<START> a salad and a sandwich <UNK> to be eaten at a restaurant <END>

虽然感觉效果还是不怎么样但是起码有句子的样子了，而且相邻的词之间有关联了。

NetworkVisualization-TensorFlow

2019年6月24日 0:36

这篇作业主要实现三种技术，分别是：saliency maps、fooling images、class visualization
用到的卷积神经网络是已经训练好的SqueezeNet，这个网络是在ImageNet上面训练的
下载这个网络的方法跟之前下coco数据集一样，打开cs231n/datasets/一一打开记事本里面的网站把数据集网络下下来就好

In [1]:
```
# As usual, a bit of setup
import time, os, json
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from cs231n.classifiers.squeezenet import SqueezeNet
from cs231n.data_utils import load_tiny_imagenet
from cs231n.image_utils import preprocess_image, deprocess_image
from cs231n.image_utils import SQUEEZENET_MEAN, SQUEEZENET_STD

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

def get_session():
    """Create a session that dynamically allocates memory."""
    # See: https://www.tensorflow.org/tutorials/using_gpu#allowing_gpu_memory_growth
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    session = tf.Session(config=config)
    return session

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

In [2]:
```
tf.reset_default_graph()
sess = get_session()

SAVE_PATH = 'cs231n/datasets/squeezenet.ckpt'
if not os.path.exists(SAVE_PATH + ".index"):
    raise ValueError("You need to download SqueezeNet!")
model = SqueezeNet(save_path=SAVE_PATH, sess=sess)
```

```
WARNING:tensorflow:From D:\anaconda\envs\cs231n\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from ten
sorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From D:\anaconda\envs\cs231n\lib\site-packages\tensorflow\python\training\saver.py:1266: checkpoint_exists (from tensorfl
ow.python.training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from cs231n/datasets/squeezenet.ckpt
```

加载squeezenet模型

In [3]:
```
from cs231n.data_utils import load_imagenet_val
X_raw, y, class_names = load_imagenet_val(num=5)

plt.figure(figsize=(12, 6))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(X_raw[i])
    plt.title(class_names[y[i]])
    plt.axis('off')
plt.gcf().tight_layout()
```

先看一下我们用的数据集

```
In [4]: X = np.array([preprocess_image(img) for img in X_raw])
```

把这些图片做预处理

下面生成一个saliency map，这个东西可以说明图片中哪些部分影响了模型对于最后那个分类label的判断

```
In [10]: def compute_saliency_maps(X, y, model):
    """
    Compute a class saliency map using the model for images X and labels y.

    Input:
    - X: Input images, numpy array of shape (N, H, W, 3)
    - y: Labels for X, numpy of shape (N,)
    - model: A SqueezeNet model that will be used to compute the saliency map.

    Returns:
    - saliency: A numpy array of shape (N, H, W) giving the saliency maps for the
    input images.
    """
    saliency = None
    # Compute the score of the correct class for each example.
    # This gives a Tensor with shape [N], the number of examples.
    #
    # Note: this is equivalent to scores[np.arange(N), y] we used in NumPy
    # for computing vectorized losses.
    correct_scores = tf.gather_nd(model.scores,
                                  tf.stack((tf.range(X.shape[0]), model.labels), axis=1))
    # tf.stack 拼接这里是每个标签前面加一个编号
    # tf.gather_nd 根据索引取值，
```

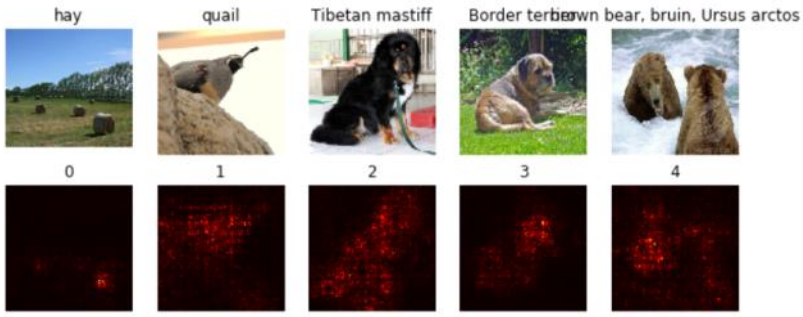
取出每个样本的正确归类的未归一化的打分

```
    ##############################################################################
    # TODO: Produce the saliency maps over a batch of images.                    #
    #                                                                            #
    # 1) Compute the "loss" using the correct scores tensor provided for you.    #
    #    (We'll combine losses across a batch by summing)                        #
    # 2) Use tf.gradients to compute the gradient of the loss with respect       #
    #    to the image (accessible via model.image).                              #
    # 3) Compute the actual value of the gradient by a call to sess.run().       #
    #    You will need to feed in values for the placeholders model.image and    #
    #    model.labels.                                                           #
    # 4) Finally, process the returned gradient to compute the saliency map.     #
    ##############################################################################
    saliency_grad = tf.gradients(correct_scores,model.image)
    # 打分对于输入图像的梯度
    saliency = sess.run(saliency_grad,{model.image:X,model.labels:y})[0]
    saliency = np.absolute(saliency)
    print(saliency.shape)
    saliency = np.amax(saliency,axis = -1)
    # 三个chanel上面的最大值
    ##############################################################################
    #                              END OF YOUR CODE                              #
    ##############################################################################
    return saliency
```

计算打分对于图片上每一个像素点的梯度，并且求梯度的绝对值在三个通道上的最大值，最终返回的是（N,H,W）

```
In [11]: def show_saliency_maps(X, y, mask):
    mask = np.asarray(mask)
    Xm = X[mask]
    ym = y[mask]

    saliency = compute_saliency_maps(Xm, ym, model)
#     print(saliency.shape)
    for i in range(mask.size):
        plt.subplot(2, mask.size, i + 1)
        plt.imshow(deprocess_image(Xm[i]))
        plt.axis('off')
        plt.title(class_names[ym[i]])
        plt.subplot(2, mask.size, mask.size + i + 1)
        plt.title(mask[i])
        plt.imshow(saliency[i], cmap=plt.cm.hot)
        plt.axis('off')
        plt.gcf().set_size_inches(10, 4)
    plt.show()

mask = np.arange(5)
show_saliency_maps(X, y, mask)
```

```
(5, 224, 224, 3)
```



这里cmap是color map ， hot表示映射成热力图

下一个是生成fooling map，这个图片跟原图片可能肉眼看上去是一样的但是让我们的分类器
判断可能会出现错误的分类。

In [24]:
```
from numpy import linalg as LA
def make_fooling_image(X, target_y, model):
    """
    Generate a fooling image that is close to X, but that the model classifies
    as target_y.

    Inputs:
    - X: Input image, a numpy array of shape (1, 224, 224, 3)
    - target_y: An integer in the range [0, 1000)
    - model: Pretrained SqueezeNet model

    Returns:
    - X_fooling: An image that is close to X, but that is classifed as target_y
    by the model.
    """

    # Make a copy of the input that we will modify
    X_fooling = X.copy()

    # Step size for the update
    learning_rate = 1
```

```
    ################################################################################
    # TODO: Generate a fooling image X_fooling that the model will classify as     #
    # the class target_y. Use gradient *ascent* on the target class score, using   #
    # the model.scores Tensor to get the class scores for the model.image.    #
    # When computing an update step, first normalize the gradient:                 #
    #   dX = learning_rate * g / ||g||_2                                           #
    #                                                                              #
    # You should write a training loop, where in each iteration, you make an       #
    # update to the input image X_fooling (don't modify X). The loop should        #
    # stop when the predicted class for the input is the same as target_y.         #
    #                                                                              #
    # HINT: It's good practice to define your TensorFlow graph operations          #
    # outside the loop, and then just make sess.run() calls in each iteration.     #
    #                                                                              #
    # HINT 2: For most examples, you should be able to generate a fooling image    #
    # in fewer than 100 iterations of gradient ascent. You can print your          #
    # progress over iterations to check your algorithm.                            #
    ################################################################################
```

```
    for i in range(100):
        target_score = tf.gather_nd(model.scores,
                                    tf.stack((tf.range(X.shape[0]),model.labels),
                                             axis = 1))
        grad_tensor = tf.gradients(target_score,model.image)
        grad,predict_logit = sess.run([grad_tensor,model.scores],
                                      {model.image:X_fooling,model.labels:[target_y]})
        cur_predict_label = np.argmax(predict_logit,axis=1)[0]
        if cur_predict_label == target_y:
            print("The labels are the same.")
            return X_fooling
        grad = grad[0]
        dx = learning_rate * grad / LA.norm(grad)

        X_fooling = X_fooling + dx
    ################################################################################
    #                              END OF YOUR CODE                                #
    ################################################################################
    return X_fooling
```

将原图片作为第一次迭代的输入，然后将你希望判别器将图片判别成的类作为target，求将图

片分为target所得到的分对图片进行求梯度，然后每次迭代将输入的图片更新，公式为：

$$dX = learning\_rate * g / ||g||_2$$

直到分类器把该图片分类到target为止

In [27]:
```
idx = 2
Xi = X[idx][None]
target_y = 6
X_fooling = make_fooling_image(Xi, target_y, model)

# Make sure that X_fooling is classified as y_target
scores = sess.run(model.scores, {model.image: X_fooling})
assert scores[0].argmax() == target_y, 'The network is not fooled!'

# Show original image, fooling image, and difference
orig_img = deprocess_image(Xi[0])
fool_img = deprocess_image(X_fooling[0])
# Rescale
plt.subplot(1, 4, 1)
plt.imshow(orig_img)
plt.axis('off')
plt.title(class_names[y[idx]])
plt.subplot(1, 4, 2)
plt.imshow(fool_img)
plt.title(class_names[target_y])
plt.axis('off')
plt.subplot(1, 4, 3)
plt.title('Difference')
plt.imshow(deprocess_image((Xi-X_fooling)[0]))
plt.axis('off')
plt.subplot(1, 4, 4)
plt.title('Magnified difference (10x)')
plt.imshow(deprocess_image(10 * (Xi-X_fooling)[0]))
plt.axis('off')
plt.gcf().tight_layout()
```

```
The labels are the same.
```

结果显示，第三张图是两个图片不同值的显示，第四张图是不同值放大了十倍的结果

最后一个是class visualization。通过产生一个随机噪声的图片，然后在目标类上做梯度上升，我们就可以生成一张模型会认为是目标类的图片了。

In [28]:
```
from scipy.ndimage.filters import gaussian_filter1d
def blur_image(X, sigma=1):
    X = gaussian_filter1d(X, sigma, axis=1)
    X = gaussian_filter1d(X, sigma, axis=2)
    return X
```

In [35]:
```
def create_class_visualization(target_y, model, **kwargs):
    """
    Generate an image to maximize the score of target_y under a pretrained model.

    Inputs:
    - target_y: Integer in the range [0, 1000) giving the index of the class
    - model: A pretrained CNN that will be used to generate the image

    Keyword arguments:
    - l2_reg: Strength of L2 regularization on the image
    - learning_rate: How big of a step to take
    - num_iterations: How many iterations to use
    - blur_every: How often to blur the image as an implicit regularizer
    - max_jitter: How much to gjitter the image as an implicit regularizer
    - show_every: How often to show the intermediate result
    """
    l2_reg = kwargs.pop('l2_reg', 1e-3)
    learning_rate = kwargs.pop('learning_rate', 25)
    num_iterations = kwargs.pop('num_iterations', 100)
    blur_every = kwargs.pop('blur_every', 10)
    max_jitter = kwargs.pop('max_jitter', 16)
    show_every = kwargs.pop('show_every', 25)

    # We use a single image of random noise as a starting point
    X = 255 * np.random.rand(224, 224, 3)
    X = preprocess_image(X)[None]
```

```
    ##############################################################################
    # TODO: Compute the loss and the gradient of the loss with respect to  #
    # the input image, model.image. We compute these outside the loop so   #
    # that we don't have to recompute the gradient graph at each iteration #
    #                                                                      #
    # Note: loss and grad should be TensorFlow Tensors, not numpy arrays!  #
    #                                                                      #
    # The loss is the score for the target label, target_y. You should     #
    # use model.scores to get the scores, and tf.gradients to compute  #
    # gradients. Don't forget the (subtracted) L2 regularization term!     #
    ##############################################################################

    loss = None # scalar loss
    grad = None # gradient of loss with respect to model.image, same size as model.image
    loss = tf.gather_nd(model.scores,
                        tf.stack((tf.range(X.shape[0]),model.labels),axis=1))
    l2_reg_tensor = tf.constant(l2_reg)
    loss = tf.subtract(loss,tf.multiply(l2_reg_tensor,tf.nn.l2_normalize(model.image,dim=[0,1,2,3,])))
    grad = tf.gradients(loss,model.image)
    ##############################################################################
    #                             END OF YOUR CODE                               #
    ##############################################################################
```

```
    for t in range(num_iterations):
        # Randomly jitter the image a bit; this gives slightly nicer results
        ox, oy = np.random.randint(-max_jitter, max_jitter+1, 2)
        X = np.roll(np.roll(X, ox, 1), oy, 2)
        # 在x方向和y方向上面随机roll
        ########################################################################
        # TODO: Use sess to compute the value of the gradient of the score for #
        # class target_y with respect to the pixels of the image, and make a   #
        # gradient step on the image using the learning rate. You should use   #
        # the grad variable you defined above.                                 #
        #                                                                      #
        # Be very careful about the signs of elements in your code.            #
        ########################################################################
        gradient = sess.run(grad,{model.image:X,model.labels:[target_y]})
        gradient = gradient[0]
        dx = learning_rate * gradient / LA.norm(gradient)
        X = X + dx
        ########################################################################
        #                             END OF YOUR CODE                         #
        ########################################################################
```

```
        # Undo the jitter
        X = np.roll(np.roll(X, -ox, 1), -oy, 2)

        # As a regularizer, clip and periodically blur
        X = np.clip(X, -SQUEEZENET_MEAN/SQUEEZENET_STD, (1.0 - SQUEEZENET_MEAN)/SQUEEZENET_STD)
        if t % blur_every == 0:
            X = blur_image(X, sigma=0.5)

        # Periodically show the image
        if t == 0 or (t + 1) % show_every == 0 or t == num_iterations - 1:
            plt.imshow(deprocess_image(X[0]))
            class_name = class_names[target_y]
            plt.title('%s\nIteration %d / %d' % (class_name, t + 1, num_iterations))
            plt.gcf().set_size_inches(4, 4)
            plt.axis('off')
            plt.show()
    return X
```

先随机生成一张图片，并做预处理
然后用下面这个公式来生成一张图片

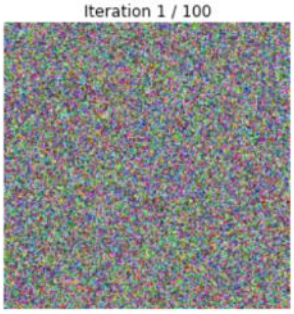
$$I^* = \arg\max_I (s_y(I) - R(I))$$

$$R(I) = \lambda \|I\|_2^2$$

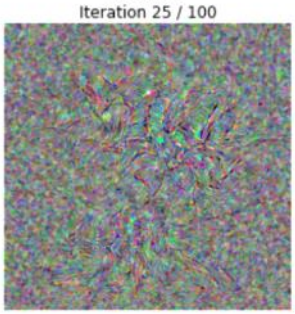
求出这个loss对图片的梯度
然后再x方向上和y方向上给图片一些随机的扰动，也就是在x方向上平移和在y方向上平移
做完更新之后再将图片逆扰动，回到原始的样子
然后再对图片做一个高斯平滑
最后看一下效果

In [36]:
```
target_y = 76 # Tarantula
out = create_class_visualization(target_y, model)
```

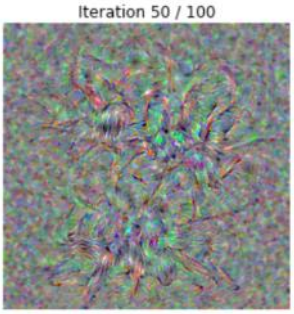
tarantula
Iteration 1 / 100



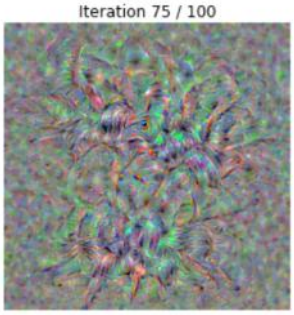
tarantula
Iteration 25 / 100

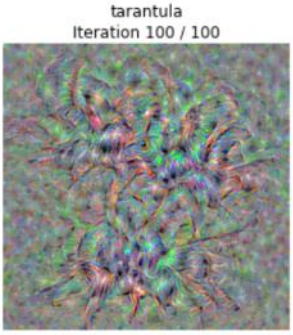


tarantula
Iteration 50 / 100



tarantula
Iteration 75 / 100

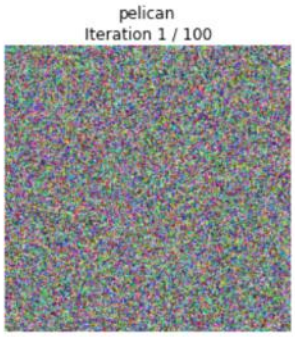


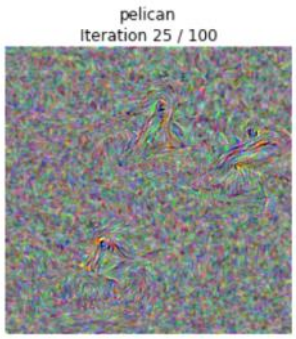


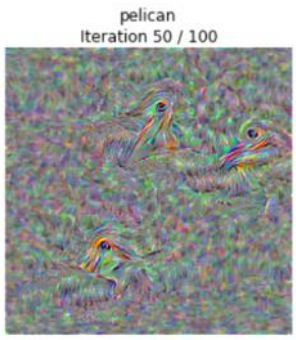
这是生成一个狼蛛照片的结果

In [37]:
```
target_y = np.random.randint(1000)
# target_y = 78 # Tick
# target_y = 187 # Yorkshire Terrier
# target_y = 683 # Oboe
# target_y = 366 # Gorilla
# target_y = 604 # Hourglass
print(class_names[target_y])
X = create_class_visualization(target_y, model)
```

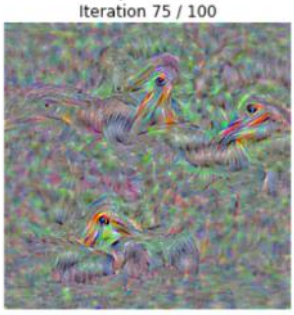
```
pelican
```



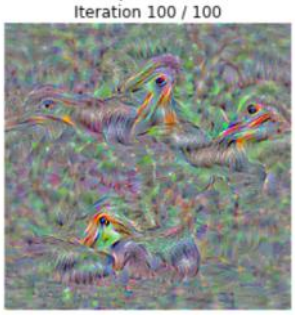




pelican
Iteration 75 / 100



pelican
Iteration 100 / 100



鹈鹕

StyleTransfer

2019年6月24日 15:22

用两张图片生成一张新的图片，新的图片包含其中一张的内容和另外一张的艺术风格

In [1]:
```
%load_ext autoreload
%autoreload 2
from scipy.misc import imread, imresize
import numpy as np

from scipy.misc import imread
import matplotlib.pyplot as plt

# Helper functions to deal with image preprocessing
from cs231n.image_utils import load_image, preprocess_image, deprocess_image

%matplotlib inline

def get_session():
    """Create a session that dynamically allocates memory."""
    # See: https://www.tensorflow.org/tutorials/using_gpu#allowing_gpu_memory_growth
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    session = tf.Session(config=config)
    return session
```

```
def rel_error(x,y):
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))

# Older versions of scipy.misc.imresize yield different results
# from newer versions, so we check to make sure scipy is up to date.
def check_scipy():
    import scipy
    version = scipy.__version__.split('.')
    if int(version[0]) < 1:
        assert int(version[1]) >= 16, "You must install SciPy >= 0.16.0 to complete this notebook."

check_scipy()
```

In [2]:
```
from cs231n.classifiers.squeezenet import SqueezeNet
import tensorflow as tf
import os

tf.reset_default_graph() # remove all existing variables in the graph
sess = get_session() # start a new Session

# Load pretrained SqueezeNet model
SAVE_PATH = 'cs231n/datasets/squeezenet.ckpt'
if not os.path.exists(SAVE_PATH + ".index"):
    raise ValueError("You need to download SqueezeNet!")
model = SqueezeNet(save_path=SAVE_PATH, sess=sess)

# Load data for testing
content_img_test = preprocess_image(load_image('styles/tubingen.jpg', size=192))[None]
style_img_test = preprocess_image(load_image('styles/starry_night.jpg', size=192))[None]
answers = np.load('style-transfer-checks-tf.npz')
```

```
WARNING:tensorflow:From D:\anaconda\envs\cs231n\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from ten
sorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From D:\anaconda\envs\cs231n\lib\site-packages\tensorflow\python\training\saver.py:1266: checkpoint_exists (from tensorfl
ow.python.training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from cs231n/datasets/squeezenet.ckpt
```

下面开始计算损失函数，损失函数分为三个部分：内容损失、风格损失还有整体多样性损失

内容损失的计算公式：

$$L_c = w_c \times \sum_{i,j}(F_{ij}^{\ell} - P_{ij}^{\ell})^2$$

F是当前图片的特征图，P是原图片的特征图，一个filter扫描整个图片得到的结果再做向量化，最终的结果就是F和P中每一行的内容，所以j等于filter的个数，i等于n*m（假设卷积核扫描后得到的结果是是一个n*m的矩阵）

In [3]:
```
def content_loss(content_weight, content_current, content_original):
    """
    Compute the content loss for style transfer.

    Inputs:
    - content_weight: scalar constant we multiply the content_loss by.
    - content_current: features of the current image, Tensor with shape [1, height, width, channels]
    - content_target: features of the content image, Tensor with shape [1, height, width, channels]

    Returns:
    - scalar content loss
    """
    return content_weight * tf.reduce_sum(tf.squared_difference(content_current,content_original))
```

In [4]:
```
def content_loss_test(correct):
    content_layer = 3
    content_weight = 6e-2
    c_feats = sess.run(model.extract_features()[content_layer], {model.image: content_img_test})
    bad_img = tf.zeros(content_img_test.shape)
    feats = model.extract_features(bad_img)[content_layer]
    student_output = sess.run(content_loss(content_weight, c_feats, feats))
    error = rel_error(correct, student_output)
    print('Maximum error is {:.3f}'.format(error))

content_loss_test(answers['cl_out'])
```

```
Maximum error is 0.000
```

下面计算风格损失，首先计算Gram矩阵，Gram矩阵可以看做feature之间的偏心协方差矩阵（即没有减去均值的协方差矩阵），可以用这个矩阵把握图片的大体风格，在这里用来衡量生成图片与风格图片的风格差异。

定义：n维欧式空间中任意$k(k \leq n)$个向量$\alpha_1, \alpha_2, \ldots, \alpha_k$的内积所组成的矩阵

$$\Delta(\alpha_1, \alpha_2, \ldots, \alpha_k) = \begin{pmatrix} (\alpha_1, \alpha_1) & (\alpha_1, \alpha_2) & \ldots & (\alpha_1, \alpha_k) \\ (\alpha_2, \alpha_1) & (\alpha_2, \alpha_2) & \ldots & (\alpha_2, \alpha_k) \\ \ldots & \ldots & \ldots & \ldots \\ (\alpha_k, \alpha_1) & (\alpha_k, \alpha_2) & \ldots & (\alpha_k, \alpha_k) \end{pmatrix}$$

称为k个向量$\alpha_1, \alpha_2, \ldots, \alpha_k$的格拉姆矩阵（Gram矩阵），它的行列式称为Gram行列式。

给定一个特征图，size为（1，C，M）C为channel数量，M是H*W，Gram矩阵的size是（1，C，C），计算公式为：

$$G_{ij}^{\ell} = \sum_k F_{ik}^{\ell} F_{jk}^{\ell}$$

设G为当前图片的Gram矩阵，A为原图片的Gram矩阵，风格损失定义为两个Gram矩阵的加权欧几里得距离

$$L_s^{\ell} = w_{\ell} \sum_{i,j} \left(G_{ij}^{\ell} - A_{ij}^{\ell} \right)^2$$

由于网络中有很多个层，我们需要计算多个层的风格损失，所以总的风格损失就是各个层的风格损失之和

$$L_s = \sum_{\ell \in \mathcal{L}} L_s^{\ell}$$

In [5]:
```
def gram_matrix(features, normalize=True):
    """
    Compute the Gram matrix from features.

    Inputs:
    - features: Tensor of shape (1, H, W, C) giving features for
      a single image.
    - normalize: optional, whether to normalize the Gram matrix
        If True, divide the Gram matrix by the number of neurons (H * W * C)

    Returns:
    - gram: Tensor of shape (C, C) giving the (optionally normalized)
      Gram matrices for the input image.
    """
    features = tf.transpose(features,[0,3,1,2]) #(1,C,H,W)
    shape = tf.shape(features)
    features = tf.reshape(features,(shape[0],shape[1],-1)) #(1,C,H*W)
    transpose_features = tf.transpose(features,[0,2,1]) #(1,H*W,C)
    result = tf.matmul(features,transpose_features)
    # 计算两个features之间的Gram矩阵
    if normalize:
        result = tf.div(result,tf.cast(shape[0]*shape[1]*shape[2]*shape[3],tf.float32))
    return result
```

如果要正则化的话，整个Gram矩阵除以元素个数H*W*C

In [6]:
```
def gram_matrix_test(correct):
    gram = gram_matrix(model.extract_features()[5])
    student_output = sess.run(gram, {model.image: style_img_test})
    error = rel_error(correct, student_output)
    print('Maximum error is {:.3f}'.format(error))

gram_matrix_test(answers['gm_out'])
```

```
WARNING:tensorflow:From <ipython-input-5-35ealed19ae7>:21: div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Maximum error is 0.000
```

In [8]:
```
def style_loss(feats, style_layers, style_targets, style_weights):
    """
    Computes the style loss at a set of layers.

    Inputs:
    - feats: list of the features at every layer of the current image, as produced by
      the extract_features function.
    - style_layers: List of layer indices into feats giving the layers to include in the
      style loss.
    - style_targets: List of the same length as style_layers, where style_targets[i] is
      a Tensor giving the Gram matrix of the source style image computed at
      layer style_layers[i].
    - style_weights: List of the same length as style_layers, where style_weights[i]
      is a scalar giving the weight for the style loss at layer style_layers[i].

    Returns:
    - style_loss: A Tensor containing the scalar style loss.
    """
    # Hint: you can do this with one for loop over the style layers, and should
    # not be very much code (~5 lines). You will need to use your gram_matrix function.
    style_losses = 0
    for i in range(len(style_layers)):
        cur_index = style_layers[i]
        cur_feat = feats[cur_index]
        cur_weight = style_weights[i]
        cur_style_target = style_targets[i]
        gramMatrix = gram_matrix(cur_feat)
        style_losses += cur_weight * tf.reduce_sum(tf.squared_difference(gramMatrix,cur_style_target))
    return style_losses
```

先根据需要计算的层的index选出该层的特征图（已经给出）然后算该特征图的Gram矩阵，
最后再跟风格图的Gram矩阵（已经给出）计算loss

In [9]:
```
def style_loss_test(correct):
    style_layers = [1, 4, 6, 7]
    style_weights = [300000, 1000, 15, 3]

    feats = model.extract_features()
    style_target_vars = []
    for idx in style_layers:
        style_target_vars.append(gram_matrix(feats[idx]))
    style_targets = sess.run(style_target_vars,
                             {model.image: style_img_test})

    s_loss = style_loss(feats, style_layers, style_targets, style_weights)
    student_output = sess.run(s_loss, {model.image: content_img_test})
    error = rel_error(correct, student_output)
    print('Error is {:.3f}'.format(error))

style_loss_test(answers['sl_out'])
```

```
Error is 0.000
```

最后一个部分是整体方差正则化，这里计算水平或者垂直方向的所有的像素对之间（水平方向上和竖直方向上相邻元素之间）的值的差的平方和作为整体的像素值的变化(total variation)。
这里对每个通道都计算一遍整体方差的正则值，再求和，最后加权算入总的loss中，计算公式为：

$$L_{tv} = w_t \times \left(\sum_{c=1}^{3}\sum_{i=1}^{H-1}\sum_{j=1}^{W}(x_{i+1,j,c}-x_{i,j,c})^2 + \sum_{c=1}^{3}\sum_{i=1}^{H}\sum_{j=1}^{W-1}(x_{i,j+1,c}-x_{i,j,c})^2\right)$$

就是遍历水平方向上和竖直方向上的相邻元素之间的平方和再求和。

In [23]:
```
def tv_loss(img, tv_weight):
    """
    Compute total variation loss.

    Inputs:
    - img: Tensor of shape (1, H, W, 3) holding an input image.
    - tv_weight: Scalar giving the weight w_t to use for the TV loss.

    Returns:
    - loss: Tensor holding a scalar giving the total variation loss
      for img weighted by tv_weight.
    """
    # Your implementation should be vectorized and not require any loops!
    shape = tf.shape(img)
    img_row_before = tf.slice(img,[0,0,0,0],[-1,shape[1]-1,-1,-1]) # (1, H, W, 3)
    img_row_after = tf.slice(img,[0,1,0,0],[-1,shape[1]-1,-1,-1])
    img_col_before = tf.slice(img,[0,0,0,0],[-1,-1,shape[2]-1,-1])
    img_col_after = tf.slice(img,[0,0,1,0],[-1,-1,shape[2]-1,-1])
    result = tv_weight * (tf.reduce_sum(tf.squared_difference(img_row_before,img_row_after))
                          + tf.reduce_sum(tf.squared_difference(img_col_before,img_col_after)))
    return result
# a = np.arange(12).reshape(1,2,2,3)
# print(a)
# a = tf.constant(a)
# print(a)
# shape = tf.shape(a)
# b = tf.slice(a,[0,0,1,0],[-1,-1,shape[2]-1,-1])
# print(sess.run(b))
```

```
[[[[ 0  1  2]
   [ 3  4  5]]

  [[ 6  7  8]
   [ 9 10 11]]]]
Tensor("Const_18:0", shape=(1, 2, 2, 3), dtype=int32)
[[[[ 3  4  5]]

  [[ 9 10 11]]]]
```

Tf.slice(要操作的矩阵，起始位置，取的值的数量)

In [24]:
```
def tv_loss_test(correct):
    tv_weight = 2e-2
    t_loss = tv_loss(model.image, tv_weight)
    student_output = sess.run(t_loss, {model.image: content_img_test})
    error = rel_error(correct, student_output)
    print('Error is {:.3f}'.format(error))

tv_loss_test(answers['tv_out'])
```

```
Error is 0.000
```

In [26]:
```
def style_transfer(content_image, style_image, image_size, style_size, content_layer, content_weight,
                   style_layers, style_weights, tv_weight, init_random = False):
    """Run style transfer!

    Inputs:
    - content_image: filename of content image
    - style_image: filename of style image
    - image_size: size of smallest image dimension (used for content loss and generated image)
    - style_size: size of smallest style image dimension
    - content_layer: layer to use for content loss
    - content_weight: weighting on content loss
    - style_layers: list of layers to use for style loss
    - style_weights: list of weights to use for each layer in style_layers
    - tv_weight: weight of total variation regularization term
    - init_random: initialize the starting image to uniform random noise
    """
    # Extract features from the content image
    content_img = preprocess_image(load_image(content_image, size=image_size))
    feats = model.extract_features(model.image)
    content_target = sess.run(feats[content_layer],
                              {model.image: content_img[None]})

    # Extract features from the style image
    style_img = preprocess_image(load_image(style_image, size=style_size))
    style_feat_vars = [feats[idx] for idx in style_layers]
    style_target_vars = []
    # Compute list of TensorFlow Gram matrices
```

```
    for style_feat_var in style_feat_vars:
        style_target_vars.append(gram_matrix(style_feat_var))
    # Compute list of NumPy Gram matrices by evaluating the TensorFlow graph on the style image
    style_targets = sess.run(style_target_vars, {model.image: style_img[None]})

    # Initialize generated image to content image

    if init_random:
        img_var = tf.Variable(tf.random_uniform(content_img[None].shape, 0, 1), name="image")
    else:
        img_var = tf.Variable(content_img[None], name="image")

    # Extract features on generated image
    feats = model.extract_features(img_var)
    # Compute loss
    c_loss = content_loss(content_weight, feats[content_layer], content_target)
    s_loss = style_loss(feats, style_layers, style_targets, style_weights)
    t_loss = tv_loss(img_var, tv_weight)
    loss = c_loss + s_loss + t_loss

    # Set up optimization hyperparameters
    initial_lr = 3.0
    decayed_lr = 0.1
    decay_lr_at = 180
    max_iter = 200
```

```
    # Create and initialize the Adam optimizer
    lr_var = tf.Variable(initial_lr, name="lr")
    # Create train_op that updates the generated image when run
    with tf.variable_scope("optimizer") as opt_scope:
        train_op = tf.train.AdamOptimizer(lr_var).minimize(loss, var_list=[img_var])
    # Initialize the generated image and optimization variables
    opt_vars = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES, scope=opt_scope.name)
    sess.run(tf.variables_initializer([lr_var, img_var] + opt_vars))
    # Create an op that will clamp the image values when run
    clamp_image_op = tf.assign(img_var, tf.clip_by_value(img_var, -1.5, 1.5))

    f, axarr = plt.subplots(1,2)
    axarr[0].axis('off')
    axarr[1].axis('off')
    axarr[0].set_title('Content Source Img.')
    axarr[1].set_title('Style Source Img.')
    axarr[0].imshow(deprocess_image(content_img))
    axarr[1].imshow(deprocess_image(style_img))
    plt.show()
    plt.figure()
```

```
    # Hardcoded handcrafted
    for t in range(max_iter):
        # Take an optimization step to update img_var
        sess.run(train_op)
        if t < decay_lr_at:
            sess.run(clamp_image_op)
        if t == decay_lr_at:
            sess.run(tf.assign(lr_var, decayed_lr))
        if t % 100 == 0:
            print('Iteration {}'.format(t))
            img = sess.run(img_var)
            plt.imshow(deprocess_image(img[0], rescale=True))
            plt.axis('off')
            plt.show()
    print('Iteration {}'.format(t))
    img = sess.run(img_var)
    plt.imshow(deprocess_image(img[0], rescale=True))
    plt.axis('off')
    plt.show()
```

In [27]:
```
# Composition VII + Tubingen
params1 = {
    'content_image' : 'styles/tubingen.jpg',
    'style_image' : 'styles/composition_vii.jpg',
    'image_size' : 192,
    'style_size' : 512,
    'content_layer' : 3,
    'content_weight' : 5e-2,
    'style_layers' : (1, 4, 6, 7),
    'style_weights' : (20000, 500, 12, 1),
    'tv_weight' : 5e-2
}

style_transfer(**params1)
```

Content Source Img.

Style Source Img.

Iteration 0

Iteration 100

Iteration 199

In [28]:
```
# Scream + Tubingen
params2 = {
    'content_image':'styles/tubingen.jpg',
    'style_image':'styles/the_scream.jpg',
    'image_size':192,
    'style_size':224,
    'content_layer':3,
    'content_weight':3e-2,
    'style_layers':[1, 4, 6, 7],
    'style_weights':[200000, 800, 12, 1],
    'tv_weight':2e-2
}

style_transfer(**params2)
```

Content Source Img.

Style Source Img.

Iteration 0

Iteration 100

Iteration 199

In [29]:
```
# Starry Night + Tubingen
params3 = {
    'content_image' : 'styles/tubingen.jpg',
    'style_image' : 'styles/starry_night.jpg',
    'image_size' : 192,
    'style_size' : 192,
    'content_layer' : 3,
    'content_weight' : 6e-2,
    'style_layers' : [1, 4, 6, 7],
    'style_weights' : [300000, 1000, 15, 3],
    'tv_weight' : 2e-2
}

style_transfer(**params3)
```

Iteration 0

Iteration 100

Iteration 199

特征反推：用训练出来的特征，生成原始图片，这里把风格损失的权重设置为0就行

In [30]:
```
# Feature Inversion -- Starry Night + Tubingen
params_inv = {
    'content_image' : 'styles/tubingen.jpg',
    'style_image' : 'styles/starry_night.jpg',
    'image_size' : 192,
    'style_size' : 192,
    'content_layer' : 3,
    'content_weight' : 6e-2,
    'style_layers' : [1, 4, 6, 7],
    'style_weights' : [0, 0, 0, 0], # we discard any contributions from style to the loss
    'tv_weight' : 2e-2,
    'init_random': True # we want to initialize our image to be random
}

style_transfer(**params_inv)
```

Content Source Img.

Style Source Img.

Iteration 0

Iteration 100

Iteration 199

最后我把model翻出来看了一下

```
with tf.variable_scope('features', reuse=reuse):
      with tf.variable_scope('layer0'):
            W = tf.get_variable("weights", shape=[3,3,3,64])
            b = tf.get_variable("bias", shape=[64])
            x = tf.nn.conv2d(x,W,[1,2,2,1],"VALID")
            x = tf.nn.bias_add(x,b)
            layers.append(x)
      with tf.variable_scope('layer1'):
            x = tf.nn.relu(x)
            layers.append(x)
      with tf.variable_scope('layer2'):
            x = tf.nn.max_pool(x,[1,3,3,1],strides=[1,2,2,1],padding='VALID')
            layers.append(x)
      with tf.variable_scope('layer3'):
            x = fire_module(x,64,16,64,64)
            layers.append(x)
      with tf.variable_scope('layer4'):
            x = fire_module(x,128,16,64,64)
            layers.append(x)
      with tf.variable_scope('layer5'):
            x = tf.nn.max_pool(x,[1,3,3,1],strides=[1,2,2,1],padding='VALID')
            layers.append(x)
      with tf.variable_scope('layer6'):
            x = fire_module(x,128,32,128,128)
            layers.append(x)
      with tf.variable_scope('layer7'):
            x = fire_module(x,256,32,128,128)
            layers.append(x)
            layers.append(x)
       with tf.variable_scope('layer8'):
            x = tf.nn.max_pool(x,[1,3,3,1],strides=[1,2,2,1],padding='VALID')
            layers.append(x)
       with tf.variable_scope('layer9'):
            x = fire_module(x,256,48,192,192)
            layers.append(x)
       with tf.variable_scope('layer10'):
            x = fire_module(x,384,48,192,192)
            layers.append(x)
       with tf.variable_scope('layer11'):
            x = fire_module(x,384,64,256,256)
            layers.append(x)
       with tf.variable_scope('layer12'):
            x = fire_module(x,512,64,256,256)
            layers.append(x)
```

一共12层，style loss需要计算的层为1,4,6,7

```
def fire_module(x,inp,sp,e11p,e33p):
      with tf.variable_scope("fire"):
            with tf.variable_scope("squeeze"):
                  W = tf.get_variable("weights",shape=[1,1,inp,sp])
                  b = tf.get_variable("bias",shape=[sp])
                  s = tf.nn.conv2d(x,W,[1,1,1,1],"VALID")+b
                  s = tf.nn.relu(s)
            with tf.variable_scope("e11"):
                  W = tf.get_variable("weights",shape=[1,1,sp,e11p])
                  b = tf.get_variable("bias",shape=[e11p])
                  e11 = tf.nn.conv2d(s,W,[1,1,1,1],"VALID")+b
                  e11 = tf.nn.relu(e11)
            with tf.variable_scope("e33"):
                  W = tf.get_variable("weights",shape=[3,3,sp,e33p])
                  b = tf.get_variable("bias",shape=[e33p])
                  e33 = tf.nn.conv2d(s,W,[1,1,1,1],"SAME")+b
                  e33 = tf.nn.relu(e33)
            return tf.concat([e11,e33],3)
```

这是squeezenet的特点fire module

Squeeze layer只使用1*1卷积核，下面的expend layer是1*1卷积核加3*3卷积核的组合

第一层是经过relu激活后的卷积层，4,6,7都是fire module层

squeezenet没有仔细看，贴一下csdn的一个博客

https://blog.csdn.net/csdnldp/article/details/78648543

主要是压缩深层卷积神经网络，用更小的CNN架构实现相同的正确率。

GAN

2019年6月24日 16:26

GAN包含一个判别器和一个生成器，判别器是传统的分类网络，是用来判断图片是真实的（在训练集中）和假的（不在训练集中）。生成器会把随机噪音作为输入，然后用一个神经网络生成一张图片，目标是让判别器以为这个图片是真的。

将训练看成最小最大博弈

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D(G(z)))\right]$$

为了优化该最小最大博弈，我们在训练中要做两件事：

1、更新生成器来最小化判别器正确的概率

2、更新判别器来最大化判别器正确的概率

在实际应用中，在更新生成器的时候使用不同的目标函数最大化判别器做出错误选择的概率，这个改变减轻了由于判别器置信度较高时导致生成器梯度消失的问题。

1、更新生成器来最大化判别器在生成数据上做出错误选择的概率

$$\max_G \mathbb{E}_{z\sim p(z)}\left[\log D(G(z))\right]$$

2、更新判别器来最大化判别器在真实数据上做出正确选择的概率

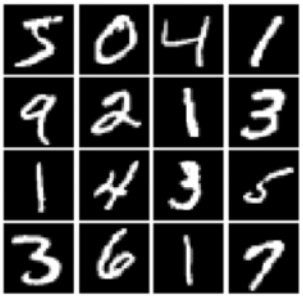
$$\max_D \mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1 - D(G(z)))\right]$$

In [1]:
```
import tensorflow as tf
import numpy as np
import os

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# A bunch of utility functions

def show_images(images):
    images = np.reshape(images, [images.shape[0], -1])  # images reshape to (batch_size, D)
    sqrtn = int(np.ceil(np.sqrt(images.shape[0])))
    sqrtimg = int(np.ceil(np.sqrt(images.shape[1])))

    fig = plt.figure(figsize=(sqrtn, sqrtn))
    gs = gridspec.GridSpec(sqrtn, sqrtn)
    gs.update(wspace=0.05, hspace=0.05)

    for i, img in enumerate(images):
        ax = plt.subplot(gs[i])
        plt.axis('off')
        ax.set_xticklabels([])
        ax.set_yticklabels([])
        ax.set_aspect('equal')
        plt.imshow(img.reshape([sqrtimg,sqrtimg]))
    return

def preprocess_img(x):
    return 2 * x - 1.0

def deprocess_img(x):
    return (x + 1.0) / 2.0

def rel_error(x,y):
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))

def count_params():
    """Count the number of parameters in the current TensorFlow graph """
    param_count = np.sum([np.prod(x.get_shape().as_list()) for x in tf.global_variables()])
    return param_count


def get_session():
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    session = tf.Session(config=config)
    return session

answers = np.load('gan-checks-tf.npz')
```

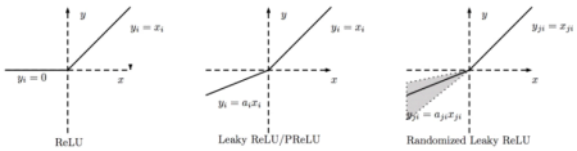
使用的数据及是MINST，每一张图片都是白色数字在黑色背景上面

In [2]:
```
class MNIST(object):
    def __init__(self, batch_size, shuffle=False):
        """
        Construct an iterator object over the MNIST data

        Inputs:
        - batch_size: Integer giving number of elements per minibatch
        - shuffle: (optional) Boolean, whether to shuffle the data on each epoch
        """
        train, _ = tf.keras.datasets.mnist.load_data()
        X, y = train
        X = X.astype(np.float32)/255
        X = X.reshape((X.shape[0], -1))
        self.X, self.y = X, y
        self.batch_size, self.shuffle = batch_size, shuffle

    def __iter__(self):
        N, B = self.X.shape[0], self.batch_size
        idxs = np.arange(N)
        if self.shuffle:
            np.random.shuffle(idxs)
        return iter((self.X[i:i+B], self.y[i:i+B]) for i in range(0, N, B))
```

In [3]:
```
# show a batch
mnist = MNIST(batch_size=16)
show_images(mnist.X[:16])
```



下面完成leaky relu
跟relu不同的是，热炉对于小于0的部分是直接舍去，然而leaky relu对于小于0 的部分是乘以
一个线性变化的非常小的比例



In [4]:
```
def leaky_relu(x, alpha=0.01):
    """Compute the leaky ReLU activation function.

    Inputs:
    - x: TensorFlow Tensor with arbitrary shape
    - alpha: leak parameter for leaky ReLU

    Returns:
    TensorFlow Tensor with the same shape as x
    """
    # TODO: implement leaky ReLU
    condition = tf.less(x,0)
    res = tf.where(condition,alpha * x,x)
    # condition中为True的替换为alpha*x，为False替换为x
    return res
```

condition加tf.where组合使用，condition找到小于0的元素的位置，tf.where把小于0的元
素乘以alpha，大于0的元素不变。

In [5]:
```
def test_leaky_relu(x, y_true):
    tf.reset_default_graph()
    with get_session() as sess:
        y_tf = leaky_relu(tf.constant(x))
        y = sess.run(y_tf)
        print('Maximum error: %g'%rel_error(y_true, y))

test_leaky_relu(answers['lrelu_x'], answers['lrelu_y'])
```

```
Maximum error: 0
```

In [6]:
```
def sample_noise(batch_size, dim):
    """Generate random uniform noise from -1 to 1.

    Inputs:
    - batch_size: integer giving the batch size of noise to generate
    - dim: integer giving the dimension of the the noise to generate

    Returns:
    TensorFlow Tensor containing uniform noise in [-1, 1] with shape [batch_size, dim]
    """
    # TODO: sample and return noise
    return tf.random_uniform([batch_size,dim],minval=-1,maxval=1)
```

这个随机噪声是[-1,1]均匀分布的

In [7]:
```
def test_sample_noise():
    batch_size = 3
    dim = 4
    tf.reset_default_graph()
    with get_session() as sess:
        z = sample_noise(batch_size, dim)
        # Check z has the correct shape
        assert z.get_shape().as_list() == [batch_size, dim]
        # Make sure z is a Tensor and not a numpy array
        assert isinstance(z, tf.Tensor)
        # Check that we get different noise for different evaluations
        z1 = sess.run(z)
        z2 = sess.run(z)
        assert not np.array_equal(z1, z2)
        # Check that we get the correct range
        assert np.all(z1 >= -1.0) and np.all(z1 <= 1.0)
        print("All tests passed!")

test_sample_noise()
```

```
All tests passed!
```

判别器:

Architecture:

- Fully connected layer with input size 784 and output size 256
- LeakyReLU with alpha 0.01
- Fully connected layer with output size 256
- LeakyReLU with alpha 0.01
- Fully connected layer with output size 1

In [8]:
```
def discriminator(x):
    """Compute discriminator score for a batch of input images.

    Inputs:
    - x: TensorFlow Tensor of flattened input images, shape [batch_size, 784]

    Returns:
    TensorFlow Tensor with shape [batch_size, 1], containing the score
    for an image being real for each input image.
    """
    with tf.variable_scope("discriminator"):
        # TODO: implement architecture
        fc1 = tf.layers.dense(x,units=256,use_bias=True,name="first_fc")
        leaky_relu1 = leaky_relu(fc1,alpha=0.01)
        fc2 = tf.layers.dense(leaky_relu1,units=256,use_bias=True,name="second_fc")
        leaky_relu2 = leaky_relu(fc2,alpha=0.01)
        logits = tf.layers.dense(leaky_relu2,units=1,name="logits")
        return logits
```

实现判别器，网络已经给出来了，照着写就行

In [9]:
```
def test_discriminator(true_count=267009):
    tf.reset_default_graph()
    with get_session() as sess:
        y = discriminator(tf.ones((2, 784)))
        cur_count = count_params()
        if cur_count != true_count:
            print('Incorrect number of parameters in discriminator. {0} instead of {1}. Check your achitecture.'.format(cur_count,true_count))
        else:
            print('Correct number of parameters in discriminator.')

test_discriminator()
```

```
WARNING:tensorflow:From <ipython-input-8-9b53639ee6ac>:13: dense (from tensorflow.python.layers.core) is deprecated and will be removed in a
future version.
Instructions for updating:
Use keras.layers.dense instead.
WARNING:tensorflow:From D:\anaconda\envs\cs231n\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from ten
sorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Correct number of parameters in discriminator.
```

生成器:

Architecture:

- Fully connected layer with inupt size tf.shape(z)[1] (the number of noise dimensions) and output size 1024
- `ReLU`
- Fully connected layer with output size 1024
- `ReLU`
- Fully connected layer with output size 784
- `TanH` (To restrict every element of the output to be in the range [-1,1])

In [10]:
```
def generator(z):
    """Generate images from a random noise vector.

    Inputs:
    - z: TensorFlow Tensor of random noise with shape [batch_size, noise_dim]

    Returns:
    TensorFlow Tensor of generated images, with shape [batch_size, 784].
    """
    with tf.variable_scope("generator"):
        # TODO: implement architecture
        fc1 = tf.layers.dense(z,units=1024,use_bias=True)
        relu1 = tf.nn.relu(fc1)
        fc2 = tf.layers.dense(relu1,units=1024,use_bias=True)
        relu2 = tf.nn.relu(fc2)
        fc3 = tf.layers.dense(relu2,units=784,use_bias=True)
        img = tf.nn.tanh(fc3)
        return img
```

In [11]:
```
def test_generator(true_count=1858320):
    tf.reset_default_graph()
    with get_session() as sess:
        y = generator(tf.ones((1, 4)))
        cur_count = count_params()
        if cur_count != true_count:
            print('Incorrect number of parameters in generator. {0} instead of {1}. Check your achitecture.'.format(cur_count,true_count))
        else:
            print('Correct number of parameters in generator.')

test_generator()
```

```
Correct number of parameters in generator.
```

下面计算GAN的loss

生成器loss:

$$\ell_G = -\mathbb{E}_{z\sim p(z)}\left[\log D(G(z))\right]$$

判别器loss:

$$\ell_D = -\mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] - \mathbb{E}_{z\sim p(z)}\left[\log(1-D(G(z)))\right]$$

In [52]:
```
def gan_loss(logits_real, logits_fake):
    """Compute the GAN loss.

    Inputs:
    - logits_real: Tensor, shape [batch_size, 1], output of discriminator
        Unnormalized score that the image is real for each real image
    - logits_fake: Tensor, shape[batch_size, 1], output of discriminator
        Unnormalized score that the image is real for each fake image

    Returns:
    - D_loss: discriminator loss scalar
    - G_loss: generator loss scalar

    HINT: for the discriminator loss, you'll want to do the averaging separately for
    its two components, and then add them together (instead of averaging once at the very end).
    """
    # TODO: compute D_loss and G_loss
    D_loss = None
    G_loss = None
    D_loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones_like(logits_real),logits=logits_real,name="discriminator_real_loss")+\
             tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.zeros_like(logits_fake),logits=logits_fake,name="discriminator_fake_loss")
    # 交叉熵越小，判断正确的概率越大
    G_loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones_like(logits_fake),logits=logits_fake,name="generator_loss")
    # 交叉熵越小，作出错误选择的概率越大
    print(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.ones_like(logits_real),logits=logits_real,name="discriminator_real_loss"))
    print(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.zeros_like(logits_fake),logits=logits_fake,name="discriminator_fake_loss"))
    D_loss = tf.reduce_mean(D_loss)
#     print(D_loss)
    G_loss = tf.reduce_mean(G_loss)
    # 求平均值
    return D_loss, G_loss
```

用sigmoid_cross_entroy_with_logits来计算loss，由于输出是batch中每一个样本的loss所
以一般配合reduce_mean使用。

In [53]:
```
def test_gan_loss(logits_real, logits_fake, d_loss_true, g_loss_true):
    tf.reset_default_graph()
    with get_session() as sess:
        d_loss, g_loss = sess.run(gan_loss(tf.constant(logits_real), tf.constant(logits_fake)))
    print("Maximum error in d_loss: %g"%rel_error(d_loss_true, d_loss))
    print("Maximum error in g_loss: %g"%rel_error(g_loss_true, g_loss))

test_gan_loss(answers['logits_real'], answers['logits_fake'],
              answers['d_loss_true'], answers['g_loss_true'])
```

```
Tensor("discriminator_real_loss_1:0", shape=(10, 1), dtype=float64)
Tensor("discriminator_fake_loss_1:0", shape=(10, 1), dtype=float64)
Maximum error in d_loss: 1.20519e-16
Maximum error in g_loss: 7.19722e-17
```

In [20]:
```
# TODO: create an AdamOptimizer for D_solver and G_solver
def get_solvers(learning_rate=1e-3, beta1=0.5):
    """Create solvers for GAN training.

    Inputs:
    - learning_rate: learning rate to use for both solvers
    - beta1: beta1 parameter for both solvers (first moment decay)

    Returns:
    - D_solver: instance of tf.train.AdamOptimizer with correct learning_rate and beta1
    - G_solver: instance of tf.train.AdamOptimizer with correct learning_rate and beta1
    """
    D_solver = None
    G_solver = None
    D_solver = tf.train.AdamOptimizer(learning_rate=learning_rate,beta1=beta1)
    G_solver = tf.train.AdamOptimizer(learning_rate=learning_rate,beta1=beta1)
    return D_solver, G_solver
```

优化用的Adam

In [66]:
```
tf.reset_default_graph()

# number of images for each batch
batch_size = 100
# our noise dimension
noise_dim = 96

# placeholder for images from the training dataset
x = tf.placeholder(tf.float32, [None, 784])
# random noise fed into our generator
z = sample_noise(batch_size, noise_dim)
# generated images
G_sample = generator(z)
# print(discriminator(G_sample))

with tf.variable_scope("") as scope:
    #scale images to be -1 to 1
    logits_real = discriminator(preprocess_img(x))
    # Re-use discriminator weights on new inputs
    scope.reuse_variables()
    logits_fake = discriminator(G_sample)
# print(logits_real)
```

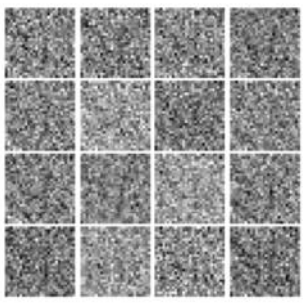
```
# Get the list of variables for the discriminator and generator
D_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, 'discriminator')
G_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, 'generator')
# print(D_vars)
# get our solver
D_solver, G_solver = get_solvers()

# get our loss
D_loss, G_loss = gan_loss(logits_real, logits_fake)
# print(tf.shape(D_loss),tf.shape(G_loss))

# setup training steps
D_train_step = D_solver.minimize(D_loss, var_list=D_vars)
# print(D_train_step)
G_train_step = G_solver.minimize(G_loss, var_list=G_vars)
D_extra_step = tf.get_collection(tf.GraphKeys.UPDATE_OPS, 'discriminator')
G_extra_step = tf.get_collection(tf.GraphKeys.UPDATE_OPS, 'generator')
```

```
Tensor("discriminator_real_loss_1:0", shape=(?, 1), dtype=float32)
Tensor("discriminator_fake_loss_1:0", shape=(100, 1), dtype=float32)
```

然后这里batch出了一些意想不到的问题，原有的128一直提示我出错，不知道为什么在梯度
更新的时候矩阵维度对应不上，但是我换成100就好了，查了半天，一直没弄清为什么会出错
.....

In [68]:
```
# tensor_name_list = [tensor.name for tensor in tf.get_default_graph().as_graph_def().node]
# for tensor_name in tensor_name_list:
#     print(tensor_name,'\n')

with get_session() as sess:
    sess.run(tf.global_variables_initializer())
    run_a_gan(sess,G_train_step,G_loss,D_train_step,D_loss,G_extra_step,D_extra_step)
```

结果



```
Epoch: 0, D: 2.738, G:0.01988
```

Epoch: 5, D: 1.217, G:0.8779

Epoch: 25, D: 1.07, G:1.183

Epoch: 45, D: 0.9778, G:1.35
Final images

差不多这样，随便截几张

下面改变损失函数
生成器：

$$\ell_G = \frac{1}{2}\mathbb{E}_{z\sim p(z)}\left[(D(G(z))-1)^2\right]$$

判别器：

$$\ell_D = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\left[(D(x)-1)^2\right] + \frac{1}{2}\mathbb{E}_{z\sim p(z)}\left[(D(G(z)))^2\right]$$

In [71]:
```
def lsgan_loss(scores_real, scores_fake):
    """Compute the Least Squares GAN loss.

    Inputs:
    - scores_real: Tensor, shape [batch_size, 1], output of discriminator
        The score for each real image
    - scores_fake: Tensor, shape[batch_size, 1], output of discriminator
        The score for each fake image

    Returns:
    - D_loss: discriminator loss scalar
    - G_loss: generator loss scalar
    """
    # TODO: compute D_loss and G_loss
    D_loss = None
    G_loss = None
    D_loss = 0.5*tf.reduce_mean(tf.square(scores_real-1))+0.5*tf.reduce_mean(tf.square(scores_fake))
    G_loss = 0.5*tf.reduce_mean(tf.square(scores_fake-1))
    return D_loss, G_loss
```

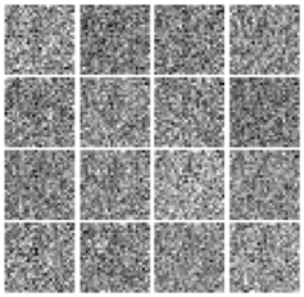
In [72]:
```
def test_lsgan_loss(score_real, score_fake, d_loss_true, g_loss_true):
    with get_session() as sess:
        d_loss, g_loss = sess.run(
            lsgan_loss(tf.constant(score_real), tf.constant(score_fake)))
    print("Maximum error in d_loss: %g"%rel_error(d_loss_true, d_loss))
    print("Maximum error in g_loss: %g"%rel_error(g_loss_true, g_loss))

test_lsgan_loss(answers['logits_real'], answers['logits_fake'],
                answers['d_loss_lsgan_true'], answers['g_loss_lsgan_true'])
```

```
Maximum error in d_loss: 0
Maximum error in g_loss: 0
```

In [73]:
```
D_loss, G_loss = lsgan_loss(logits_real, logits_fake)
D_train_step = D_solver.minimize(D_loss, var_list=D_vars)
G_train_step = G_solver.minimize(G_loss, var_list=G_vars)
```

In [74]:
```
with get_session() as sess:
    sess.run(tf.global_variables_initializer())
    run_a_gan(sess, G_train_step, G_loss, D_train_step, D_loss, G_extra_step, D_extra_step)
```

```
100
```



```
Epoch: 45, D: 0.1736, G:0.2106
Final images
```

这个结果更好一些

下面来看Deep convolutional GAN
使用卷积层之后，网络能形成真实的空间上的特征，可以生成出“锋利的边缘”
判别器：

Architecture:

- Conv2D: 32 Filters, 5x5, Stride 1, padding 0
- Leaky ReLU(alpha=0.01)
- Max Pool 2x2, Stride 2
- Conv2D: 64 Filters, 5x5, Stride 1, padding 0
- Leaky ReLU(alpha=0.01)
- Max Pool 2x2, Stride 2
- Flatten
- Fully Connected with output size 4 x 4 x 64
- Leaky ReLU(alpha=0.01)
- Fully Connected with output size 1

In [77]:
```
def discriminator(x):
    """Compute discriminator score for a batch of input images.

    Inputs:
    - x: TensorFlow Tensor of flattened input images, shape [batch_size, 784]

    Returns:
    TensorFlow Tensor with shape [batch_size, 1], containing the score
    for an image being real for each input image.
    """
    x = tf.reshape(x, shape=(tf.shape(x)[0], 28, 28, 1))
    with tf.variable_scope("discriminator"):
        # TODO: implement architecture
        conv1 = tf.layers.conv2d(x, filters=32, kernel_size=(5, 5), strides=(1, 1), activation=leaky_relu)
        max_pool1 = tf.layers.max_pooling2d(conv1, pool_size=(2, 2), strides=(2, 2))
        conv2 = tf.layers.conv2d(max_pool1, filters=64, kernel_size=(5, 5), strides=(1, 1), activation=leaky_relu)
        max_pool2 = tf.layers.max_pooling2d(conv2, pool_size=(2, 2), strides=(2, 2))
        flat = tf.contrib.layers.flatten(max_pool2)
        fc1 = tf.layers.dense(flat, units=4*4*64, activation=leaky_relu)
        logits = tf.layers.dense(fc1, units=1)
        return logits
test_discriminator(1102721)
```

```
WARNING:tensorflow:From <ipython-input-77-dd0c1dd61a6b>:15: max_pooling2d (from tensorflow.python.layers.pooling) is deprecated and will be
removed in a future version.
Instructions for updating:
Use keras.layers.max_pooling2d instead.

WARNING: The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From D:\anaconda\envs\cs231n\lib\site-packages\tensorflow\contrib\layers\python\layers\layers.py:1624: flatten (from tens
orflow.python.layers.core) is deprecated and will be removed in a future version.
Instructions for updating:
Use keras.layers.flatten instead.
Correct number of parameters in discriminator.
```

生成器:

Architecture:

- Fully connected with output size 1024
- `ReLU`
- BatchNorm
- Fully connected with output size 7 x 7 x 128
- `ReLU`
- BatchNorm
- Resize into Image Tensor of size 7, 7, 128
- Conv2D^T (transpose): 64 filters of 4x4, stride 2
- `ReLU`
- BatchNorm
- Conv2d^T (transpose): 1 filter of 4x4, stride 2
- `TanH`

In [81]:
```
def generator(z):
    """Generate images from a random noise vector.

    Inputs:
    - z: TensorFlow Tensor of random noise with shape [batch_size, noise_dim]

    Returns:
    TensorFlow Tensor of generated images, with shape [batch_size, 784].
    """
    batch_size = tf.shape(z)[0]
    with tf.variable_scope("generator"):
        # TODO: implement architecture
        fc1 = tf.layers.dense(z, units=1024, activation=tf.nn.relu, use_bias=True)
        bn1 = tf.layers.batch_normalization(fc1, training=True)
        fc2 = tf.layers.dense(bn1, units=7*7*128, activation=tf.nn.relu, use_bias=True)
        bn2 = tf.layers.batch_normalization(fc2, training=True)
        resize = tf.reshape(bn2, shape=(-1, 7, 7, 128))
        filter_conv1 = tf.get_variable('decov1', [4, 4, 64, 128]) #height weight output in
        conv_tr1 = tf.nn.conv2d_transpose(resize, filter=filter_conv1, output_shape=[batch_size, 14, 14, 64], strides=[1, 2, 2, 1])
        bias1 = tf.get_variable('deconv1_bias', [64])
        conv_tr1 = conv_tr1 + bias1
        relu_conv_tr1 = tf.nn.relu(conv_tr1)
        bn3 = tf.layers.batch_normalization(relu_conv_tr1, training=True)
        filter_conv2 = tf.get_variable('deconv2', [4, 4, 1, 64])
        conv_tr2 = tf.nn.conv2d_transpose(bn3, filter=filter_conv2, output_shape=[batch_size, 28, 28, 1], strides=[1, 2, 2, 1])
        bias2 = tf.get_variable('deconv2_bias', [1])
        conv_tr2 = conv_tr2 + bias2
        img = tf.nn.tanh(conv_tr2)
        img = tf.contrib.layers.flatten(img)
        return img
```

In [84]:
```
tf.reset_default_graph()

batch_size = 100
# our noise dimension
noise_dim = 96

# placeholders for images from the training dataset
x = tf.placeholder(tf.float32, [None, 784])
z = sample_noise(batch_size, noise_dim)
# 生成一个batch_size,dim的噪声，用于后面生成img
# generated images
G_sample = generator(z)
# 生成器生成img

with tf.variable_scope("") as scope:
    #scale images to be -1 to 1
    logits_real = discriminator(preprocess_img(x))
    # Re-use discriminator weights on new inputs
    scope.reuse_variables()
    # 重用变量
    logits_fake = discriminator(G_sample)
    # 判别器判断
```

```
# Get the list of variables for the discriminator and generator
D_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,'discriminator')
G_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,'generator')
# 获取生成器和判别器的可训练变量

D_solver,G_solver = get_solvers()
# 生成优化器
D_loss, G_loss = gan_loss(logits_real, logits_fake)
# 计算loss
D_train_step = D_solver.minimize(D_loss, var_list=D_vars)
G_train_step = G_solver.minimize(G_loss, var_list=G_vars)
D_extra_step = tf.get_collection(tf.GraphKeys.UPDATE_OPS,'discriminator')
G_extra_step = tf.get_collection(tf.GraphKeys.UPDATE_OPS,'generator')
```

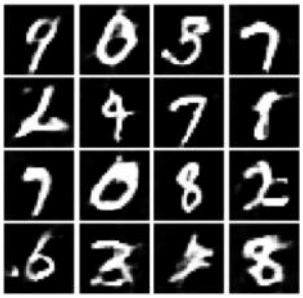
```
Tensor("discriminator_real_loss_1:0", shape=(?, 1), dtype=float32)
Tensor("discriminator_fake_loss_1:0", shape=(100, 1), dtype=float32)
```

还是同样的问题，batch_size还是要改

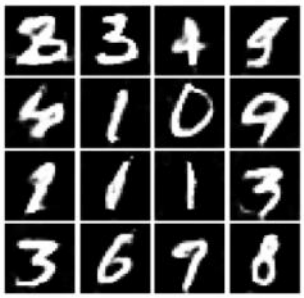
In [87]:
```
with get_session() as sess:
    sess.run(tf.global_variables_initializer())
    run_a_gan(sess,G_train_step,G_loss,D_train_step,D_loss,G_extra_step,D_extra_step,num_epoch=5)
```

```
100
```

```
Epoch: 0, D: 0.7897, G:1.535
```

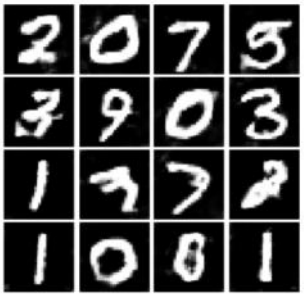


```
Epoch: 1, D: 0.7355, G:1.06
```



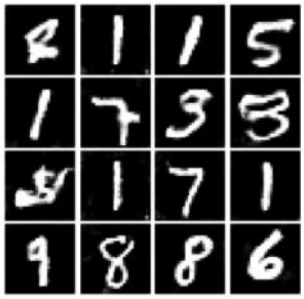
Epoch: 2, D: 0.7961, G:1.035

Epoch: 3, D: 0.8791, G:1.322



Epoch: 4, D: 0.8465, G:1.271

Final images



显示最终结果