In [ ]:

In [11]:
```
# As usual, a bit of setup

from __future__ import print_function
import time, os, json
import numpy as np
import matplotlib.pyplot as plt

from deeplearning.gradient_check import eval_numerical_gradient, eval
_numerical_gradient_array
from deeplearning.rnn_layers import *
from deeplearning.captioning_solver import CaptioningSolver
from deeplearning.classifiers.rnn import CaptioningRNN
from deeplearning.coco_utils import load_coco_data, sample_coco_minib
atch, decode_captions
from deeplearning.image_utils import image_from_url

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

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-module
s-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.ab
s(y))))
```

Install h5py

The COCO dataset we will be using is stored in HDF5 format. To load HDF5 files, we will need to install the `h5py` Python package. From the command line, run:

```
pip install h5py
```

If you receive a permissions error, you may need to run the command as root:

```
sudo pip install h5py
```

You can also run commands directly from the Jupyter notebook by prefixing the command with the "!" character:

In [10]:
```
!pip install h5py
```

```
Requirement already satisfied: h5py in ./.env/lib/python3.6/site-pack
ages (2.7.0)
Requirement already satisfied: six in ./.env/lib/python3.6/site-packa
ges (from h5py) (1.10.0)
Requirement already satisfied: numpy>=1.7 in ./.env/lib/python3.6/sit
e-packages (from h5py) (1.12.1)
```

In [12]:
```
# 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
```

In [13]:
```
# Sample a minibatch and show the images and captions
batch_size = 3

captions, features, urls = sample_coco_minibatch(data, batch_size=bat
ch_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 woman wearing a short <UNK> kneeling on a tennis court <END>

<START> a tour bus driving through an empty intersection <END>

<START> a person skiing on skis on a snow covered hill <END>

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

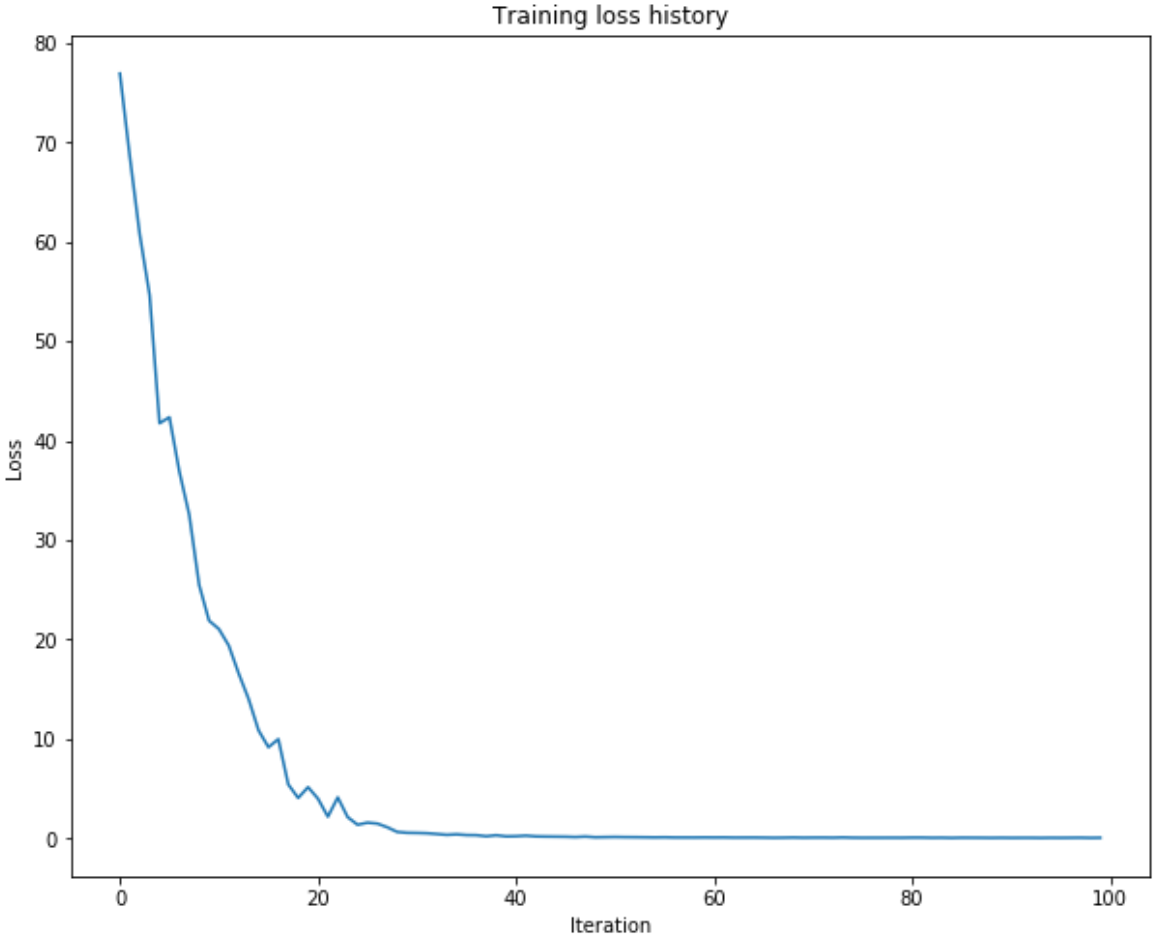
In [ ]:

In [ ]:

In [ ]:

In [24]:
```
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: 21.062919
(Iteration 21 / 100) loss: 4.016281
(Iteration 31 / 100) loss: 0.566975
(Iteration 41 / 100) loss: 0.239450
(Iteration 51 / 100) loss: 0.161949
(Iteration 61 / 100) loss: 0.111534
(Iteration 71 / 100) loss: 0.097569
(Iteration 81 / 100) loss: 0.099074
(Iteration 91 / 100) loss: 0.073962
```



Test-time sampling

In [37]:
```
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_w
ord'])

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

train
a closeup of a woman from the <UNK> to <UNK> <END>
GT:<START> a closeup of a woman from the <UNK> to <UNK> <END>

train
a man standing on the side of a road with bags of luggage <END>
GT:<START> a man standing on the side of a road with bags of luggage <END>

val
sitting atop the pile of skiing gear in room <UNK> with wood <END>
GT:<START> a man looks <UNK> and holds up his kite <END>

val
a dog is on the <END>
GT:<START> a white bed in the middle of a home room <END>

In [ ]:

In [ ]: