Image Captioning with LSTMs

在上一个练习中，您实现了一个vanillarnn并将其应用于图像描述。在本笔记本中，您将实现LSTM更新规则并将其用于图像描述。

In [2]:
```
# Setup cell.
import time, os, json
import numpy as np
import matplotlib.pyplot as plt

from daseCV.gradient_check import eval_numerical_gradient, eval_numerical_gradient_ar
from daseCV.rnn_layers import *
from daseCV.captioning_solver import CaptioningSolver
from daseCV.classifiers.rnn import CaptioningRNN
from daseCV.coco_utils import load_coco_data, sample_coco_minibatch, decode_captions
from daseCV.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'

%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))))
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

COCO Dataset

与上一个notebook一样，我们将使用COCO数据集来显示描述。

In [3]:
```
# Load COCO data from disk into a dictionary.
data = load_coco_data(base_dir="./input/datasets/coco_captioning",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))
```

```
base dir  ./input/datasets/coco_captioning
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
```

LSTM

普通RNN的一个常见变体是长短时记忆（LSTM）RNN。由于重复矩阵乘法引起的梯度消失和爆炸，普通RNN很难在长序列上训练。LSTMs通过用如下的选通机制替换vanillarnn的简单更新规则来解决这个问题。

Similar to the vanilla RNN, at each timestep we receive an input $x_t \in \mathbb{R}^D$ and the previous hidden state $h_{t-1} \in \mathbb{R}^H$; the LSTM also maintains an H-dimensional *cell state*, so we also receive the previous cell state $c_{t-1} \in \mathbb{R}^H$. The learnable parameters of the LSTM are an *input-to-hidden* matrix $W_x \in \mathbb{R}^{4H \times D}$, a *hidden-to-hidden* matrix $W_h \in \mathbb{R}^{4H \times H}$ and a *bias vector* $b \in \mathbb{R}^{4H}$.

At each timestep we first compute an *activation vector* $a \in \mathbb{R}^{4H}$ as $a = W_x x_t + W_h h_{t-1} + b$. We then divide this into four vectors $a_i, a_f, a_o, a_g \in \mathbb{R}^H$ where a_i consists of the first H elements of a, a_f is the next H elements of a, etc. We then compute the *input gate* $g \in \mathbb{R}^H$, *forget gate* $f \in \mathbb{R}^H$, *output gate* $o \in \mathbb{R}^H$ and *block input* $g \in \mathbb{R}^H$ as

$$i = \sigma(a_i) \qquad f = \sigma(a_f) \qquad o = \sigma(a_o) \qquad g = \tanh(a_g)$$

where σ is the sigmoid function and $\tanh$ is the hyperbolic tangent, both applied elementwise.

Finally we compute the next cell state c_t and next hidden state h_t as

$$c_t = f \odot c_{t-1} + i \odot g \qquad h_t = o \odot \tanh(c_t)$$

where $\odot$ is the elementwise product of vectors.

In the rest of the notebook we will implement the LSTM update rule and apply it to the image captioning task.

In the code, we assume that data is stored in batches so that $X_t \in \mathbb{R}^{N \times D}$ and will work with *transposed* versions of the parameters: $W_x \in \mathbb{R}^{D \times 4H}$, $W_h \in \mathbb{R}^{H \times 4H}$ so that activations $A \in \mathbb{R}^{N \times 4H}$ can be computed efficiently as $A = X_t W_x + H_{t-1} W_h$

LSTM: Step Forward

Implement the forward pass for a single timestep of an LSTM in the `lstm_step_forward` function in the file `daseCV/rnn_layers.py`. This should be similar to the `rnn_step_forward` function that you implemented above, but using the LSTM update rule instead.

Once you are done, run the following to perform a simple test of your implementation. You should see errors on the order of `e-8` or less.

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:  5.7054131967097955e-09
next_c error:  5.8143123088804145e-09
```

LSTM: Step Backward

在函数中为单个LSTM timestep实现向后传递，如在文件 daseCV/rnn_layers.py 中 lstm_step_backward 一样。完成后，运行以下命令对实现执行数值渐变检查。你应该看到错误的顺序为 e-7 或更少

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:  6.335032254429549e-10
dh error:  3.3963774090592634e-10
dc error:  1.5221723979041107e-10
dWx error:  2.1010960934639614e-09
dWh error:  9.712296109943072e-08
db error:  2.491522041931035e-10
```

LSTM: Forward

在文件 daseCV/rnn_layers.py 中的 lstm_forward 函数中，实现 lstm_forward 函数以在整个数据时间序列上运行LSTM forward。

完成后，运行以下命令检查实现。你应该看到一个错误的顺序为 e-7 或更少。

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
```

LSTM: Backward

在 lstm_backward 函数中的整个时间序列数据上实现LSTM的向后传递，如在文件 daseCV/rnn_layers.py 中一样。完成后，运行以下命令对实现执行数值渐变检查。你应该看到错误的顺序为 e-8 或更少(对于 dWh ，如果您的错误在 e-6 或更小的顺序上，则比较好)。

In [8]:

```
from daseCV.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:  6.9939005453315376e-09
dh0 error:  1.0363303343006366e-09
dWx error:  3.2262956411424662e-09
dWh error:  2.6984652580094597e-06
db error:  8.359849381888026e-10
```

LSTM Captioning Model

现在已经实现了LSTM，请更新文件 daseCV/classifiers/rnn.py 中 CaptioningRNN 类的 loss 方法的实现，以处理 self.cell_type 是 lstm 的情况。这需要添加少于10行的代码。

完成后，运行以下命令检查实现。您应该看到 e-10 或更小的顺序上的差异。

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:  9.824459354432264
expected loss:  9.82445935443
difference:  2.2648549702353193e-12
```

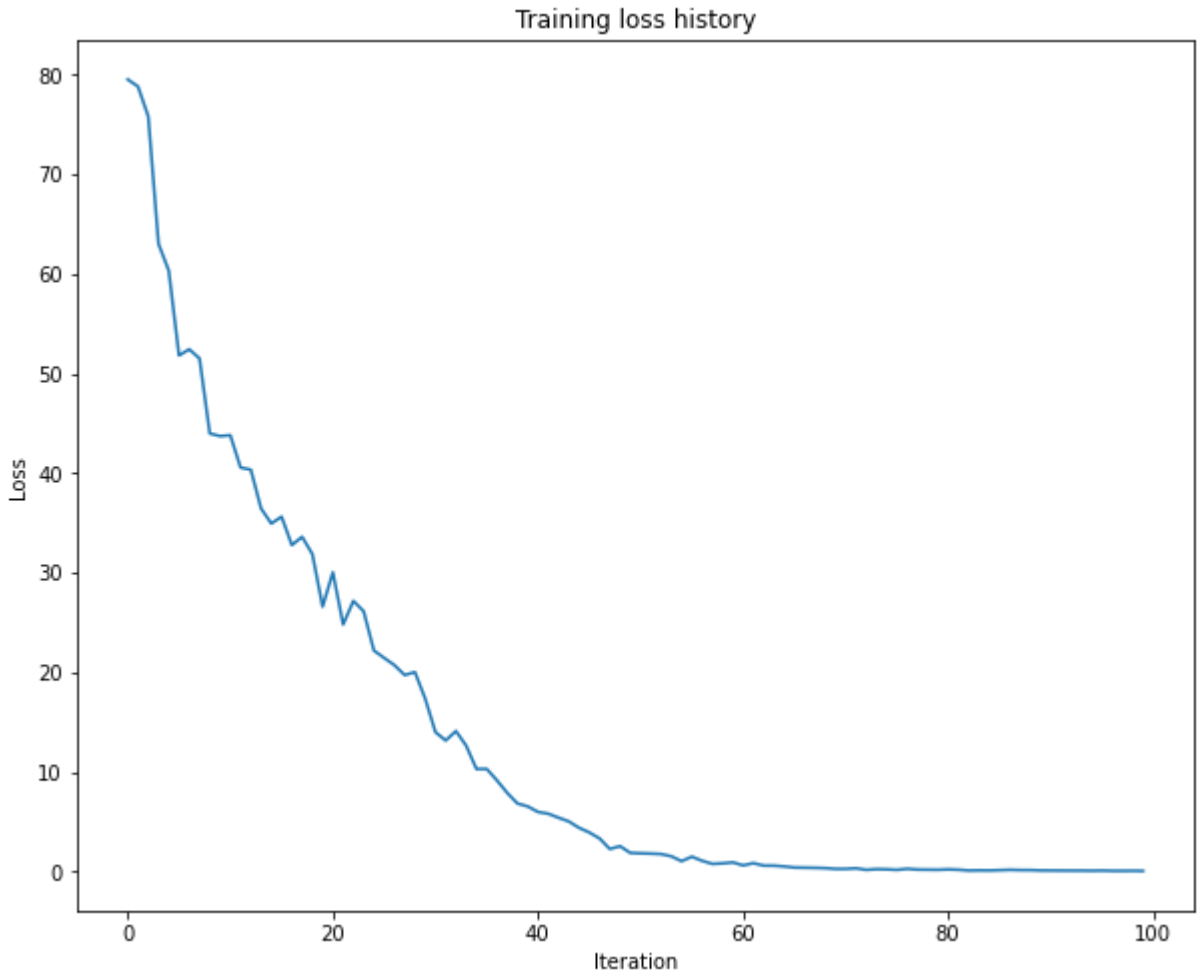
Overfit LSTM Captioning Model on Small Data

运行以下命令，在与我们之前用于RNN的相同的小数据集上过度拟合LSTM描述模型。最终损失应小于0.5。

In [11]:

```
np.random.seed(231)
small_data = load_coco_data(base_dir="./input/datasets/coco_captioning",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()
```

```
base dir  ./input/datasets/coco_captioning
(Iteration 1 / 100) loss: 79.551150
(Iteration 11 / 100) loss: 43.829101
(Iteration 21 / 100) loss: 30.062625
(Iteration 31 / 100) loss: 14.020167
(Iteration 41 / 100) loss: 6.004904
(Iteration 51 / 100) loss: 1.851745
(Iteration 61 / 100) loss: 0.638921
(Iteration 71 / 100) loss: 0.286683
(Iteration 81 / 100) loss: 0.236246
(Iteration 91 / 100) loss: 0.124038
```



Print final training loss. You should see a final loss of less than 0.5.

In [12]:
```
print('Final loss: ', small_lstm_solver.loss_history[-1])
```

```
Final loss:  0.08100990611519694
```

LSTM Sampling at Test Time

修改 CaptioningRNN 类的 sample 方法以处理 self.cell_type 是 lstm 的情况。这应该需要少于10行代码。

完成后，在一些训练集和验证集样本上运行下面的示例，从过拟合LSTM模型中获取样本。与RNN一样，训练结果应该非常好，验证结果可能没有多大意义（因为我们的拟合过度了）。

In [13]:
```
# If you get an error, the URL just no longer exists, so don't worry!
# You can re-sample as many times as you want.
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):
        img = image_from_url(url)
        # Skip missing URLs.
        if img is None: continue
```

```
        plt.imshow(img)
        plt.title('%s\n%s\nGT:%s' % (split, sample_caption, gt_caption))
        plt.axis('off')
        plt.show()
```

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>

train
a man <UNK> with a bright colorful kite <END>
GT:<START> a man <UNK> with a bright colorful kite <END>

val
a person <UNK> of a <UNK> <END>
GT:<START> a sign that is on the front of a train station <END>

val
a cat is <UNK> and a <UNK> <END>
GT:<START> a car is parked on a street at night <END>

In [ ]: