# LSTM_Captioning

November 8, 2018

## 1 Image Captioning with LSTMs

In the previous exercise you implemented a vanilla RNN and applied it to image captioning. In this notebook you will implement the LSTM update rule and use it for image captioning.

```
In [2]: # 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 cs231n.gradient_check import eval_numerical_gradient, eval_numerical_gradient_arra
        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))))
```

## 2 Load MS-COCO data

As in the previous notebook, we will use the Microsoft COCO dataset for captioning.

```
In [3]: # 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))
```

```
idx_to_word <type 'list'> 1004
train_captions <type 'numpy.ndarray'> (400135, 17) int32
val_captions <type 'numpy.ndarray'> (195954, 17) int32
train_image_idxs <type 'numpy.ndarray'> (400135,) int32
val_features <type 'numpy.ndarray'> (40504, 512) float32
val_image_idxs <type 'numpy.ndarray'> (195954,) int32
train_features <type 'numpy.ndarray'> (82783, 512) float32
train_urls <type 'numpy.ndarray'> (82783,) |S63
val_urls <type 'numpy.ndarray'> (40504,) |S63
word_to_idx <type 'dict'> 1004
```

## 3 LSTM

If you read recent papers, you'll see that many people use a variant on the vanialla RNN called Long-Short Term Memory (LSTM) RNNs. Vanilla RNNs can be tough to train on long sequences due to vanishing and exploding gradiants caused by repeated matrix multiplication. LSTMs solve this problem by replacing the simple update rule of the vanilla RNN with a gating mechanism as follows.

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 $\sigma$ 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

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

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 $A = X_t W_x + H_{t-1} W_h$

# 4 LSTM: step forward

Implement the forward pass for a single timestep of an LSTM in the `lstm_step_forward` function in the file `cs231n/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 around `1e-8` or less.

```
In [4]: 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:  5.7054130404539434e-09
next_c error:  5.8143123088804145e-09
```

# 5 LSTM: step backward

Implement the backward pass for a single LSTM timestep in the function `lstm_step_backward` in the file `cs231n/rnn_layers.py`. Once you are done, run the following to perform numeric gradient checking on your implementation. You should see errors around `1e-6` or less.

```
In [5]: np.random.seed(231)
```

```python
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:  7.481575302257938e-10
dh error:  2.982650711677508e-10
dc error:  7.650761924704044e-11
```

```
dWx error:  2.3114133245750027e-09
dWh error:  9.799799942884514e-08
db error:  2.747391209539675e-10
```

## 6 LSTM: forward

In the function `lstm_forward` in the file `cs231n/rnn_layers.py`, implement the `lstm_forward` function to run an LSTM forward on an entire timeseries of data.

When you are done, run the following to check your implementation. You should see an error around `1e-7`.

```
In [6]: 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
```

## 7 LSTM: backward

Implement the backward pass for an LSTM over an entire timeseries of data in the function `lstm_backward` in the file `cs231n/rnn_layers.py`. When you are done, run the following to perform numeric gradient checking on your implementation. You should see errors around `1e-7` or less.

```
In [7]: from cs231n.rnn_layers import lstm_forward, lstm_backward
        np.random.seed(232)

        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:  6.934610940788449e-10
dh0 error:  1.7191742787512714e-10
dWx error:  3.513538003903274e-09
dWh error:  5.230244747647281e-08
db error:  6.191166792854841e-10
```

## 8 LSTM captioning model

Now that you have implemented an LSTM, update the implementation of the `loss` method of the `CaptioningRNN` class in the file `cs231n/classifiers/rnn.py` to handle the case where `self.cell_type` is `lstm`. This should require adding less than 10 lines of code.

Once you have done so, run the following to check your implementation. You should see a difference of less than `1e-10`.

```
In [8]: 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:  9.824459354432268
expected loss:  9.82445935443
difference:  2.2684076839141 2e-12
```

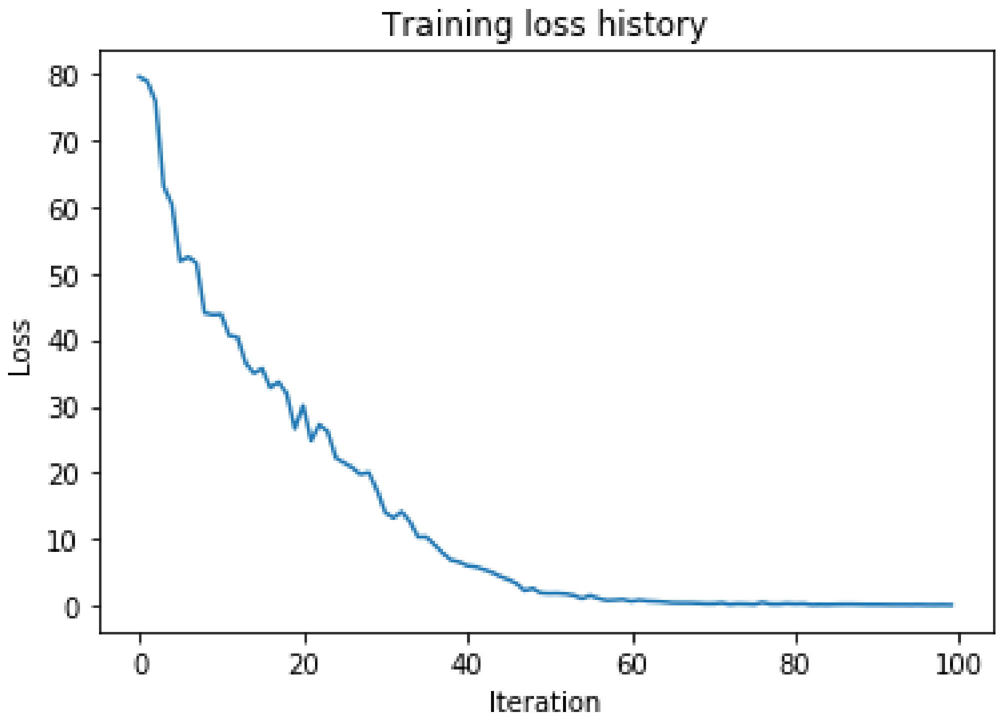
# 9 Overfit LSTM captioning model

Run the following to overfit an LSTM captioning model on the same small dataset as we used for the RNN previously. You should see losses less than 0.5.

```
In [9]: 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()
```

```
(Iteration 1 / 100) loss: 79.551150
(Iteration 11 / 100) loss: 43.829085
(Iteration 21 / 100) loss: 30.062635
(Iteration 31 / 100) loss: 14.019562
(Iteration 41 / 100) loss: 5.993702
(Iteration 51 / 100) loss: 1.837746
(Iteration 61 / 100) loss: 0.651672
(Iteration 71 / 100) loss: 0.283533
(Iteration 81 / 100) loss: 0.248227
(Iteration 91 / 100) loss: 0.154856
```




## 10 LSTM test-time sampling

Modify the `sample` method of the `CaptioningRNN` class to handle the case where `self.cell_type` is `lstm`. This should take fewer than 10 lines of code.

When you are done run the following to sample from your overfit LSTM model on some training and validation set samples.

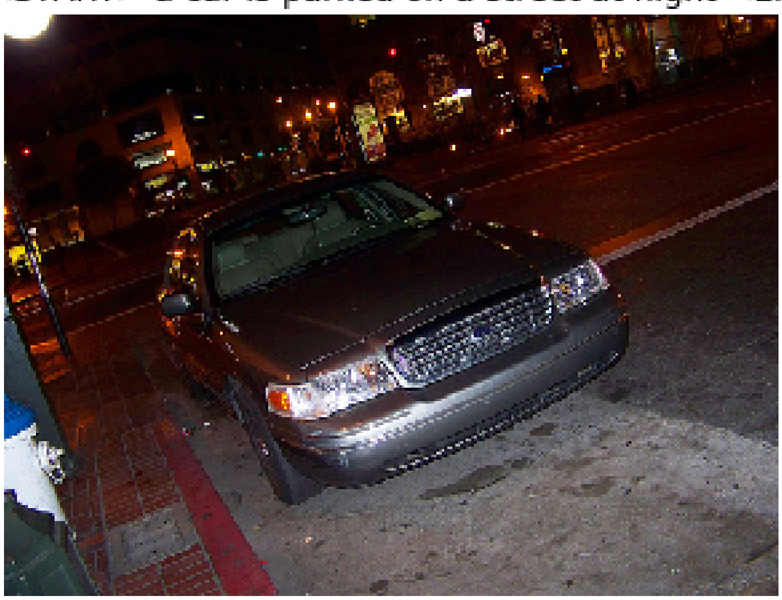
```
In [10]: for split in ['train', 'val']:
             minibatch = sample_coco_minibatch(small_data, split=split, batch_size=2)
             gt_captions, features, urls = minibatch
             gt_captions = decode_captions(gt_captions, data['idx_to_word'])

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


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



## 11 Extra Credit: Train a good captioning model!

Using the pieces you have implemented in this and the previous notebook, try to train a captioning model that gives decent qualitative results (better than the random garbage you saw with the overfit models) when sampling on the validation set. You can subsample the training set if you want; we just want to see samples on the validation set that are better than random.

In addition to qualitatively evaluating your model by inspecting its results, you can also quantitatively evaluate your model using the BLEU unigram precision metric. We'll give you a small amount of extra credit if you can train a model that achieves a BLEU unigram score of >0.3. BLEU scores range from 0 to 1; the closer to 1, the better. Here's a reference to the paper that introduces BLEU if you're interested in learning more about how it works.

Feel free to use PyTorch for this section if you'd like to train faster on a GPU… though you can definitely get above 0.3 using your Numpy code. We're providing you the evaluation code that is compatible with the Numpy model as defined above… you should be able to adapt it for PyTorch if you go that route.

```
In [11]: def BLEU_score(gt_caption, sample_caption):
             """
             gt_caption: string, ground-truth caption
             sample_caption: string, your model's predicted caption
             Returns unigram BLEU score.
             """
             reference = [x for x in gt_caption.split(' ')
                          if ('<END>' not in x and '<START>' not in x and '<UNK>' not in x)]
             hypothesis = [x for x in sample_caption.split(' ')
                           if ('<END>' not in x and '<START>' not in x and '<UNK>' not in x)]
             BLEUscore = nltk.translate.bleu_score.sentence_bleu([reference], hypothesis, weig
             return BLEUscore

         def evaluate_model(model):
             """
             model: CaptioningRNN model
             Prints unigram BLEU score averaged over 1000 training and val examples.
             """
             for split in ['train', 'val']:
                 minibatch = sample_coco_minibatch(med_data, split=split, batch_size=1000)
                 gt_captions, features, urls = minibatch
                 gt_captions = decode_captions(gt_captions, data['idx_to_word'])

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

                 total_score = 0.0
                 for gt_caption, sample_caption, url in zip(gt_captions, sample_captions, urls
```

```
                total_score += BLEU_score(gt_caption, sample_caption)

            BLEUscores[split] = total_score / len(sample_captions)

        for split in BLEUscores:
            print('Average BLEU score for %s: %f' % (split, BLEUscores[split]))
```

In [12]:
```
med_data = load_coco_data(max_train=5000)

capt_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,
        )

capt_solver = CaptioningSolver(capt_model, med_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,
         )

capt_solver.train()

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

```
(Iteration 1 / 10000) loss: 74.899128
(Iteration 11 / 10000) loss: 56.302317
(Iteration 21 / 10000) loss: 50.045852
(Iteration 31 / 10000) loss: 45.122789
(Iteration 41 / 10000) loss: 45.483515
(Iteration 51 / 10000) loss: 44.553324
(Iteration 61 / 10000) loss: 42.492130
(Iteration 71 / 10000) loss: 41.183786
(Iteration 81 / 10000) loss: 37.419331
(Iteration 91 / 10000) loss: 38.891199
```

```
(Iteration 101 / 10000) loss: 39.540272
(Iteration 111 / 10000) loss: 37.408784
(Iteration 121 / 10000) loss: 35.144494
(Iteration 131 / 10000) loss: 35.352903
(Iteration 141 / 10000) loss: 33.034799
(Iteration 151 / 10000) loss: 32.667663
(Iteration 161 / 10000) loss: 34.078115
(Iteration 171 / 10000) loss: 34.984182
(Iteration 181 / 10000) loss: 35.662743
(Iteration 191 / 10000) loss: 34.972867
(Iteration 201 / 10000) loss: 31.820422
(Iteration 211 / 10000) loss: 31.366926
(Iteration 221 / 10000) loss: 30.145294
(Iteration 231 / 10000) loss: 31.713844
(Iteration 241 / 10000) loss: 36.350150
(Iteration 251 / 10000) loss: 33.716431
(Iteration 261 / 10000) loss: 27.097244
(Iteration 271 / 10000) loss: 30.186552
(Iteration 281 / 10000) loss: 34.436102
(Iteration 291 / 10000) loss: 31.438949
(Iteration 301 / 10000) loss: 28.582804
(Iteration 311 / 10000) loss: 29.053929
(Iteration 321 / 10000) loss: 28.696845
(Iteration 331 / 10000) loss: 27.629697
(Iteration 341 / 10000) loss: 28.118037
(Iteration 351 / 10000) loss: 28.799816
(Iteration 361 / 10000) loss: 31.801705
(Iteration 371 / 10000) loss: 28.426097
(Iteration 381 / 10000) loss: 29.373626
(Iteration 391 / 10000) loss: 27.650295
(Iteration 401 / 10000) loss: 26.229482
(Iteration 411 / 10000) loss: 31.830561
(Iteration 421 / 10000) loss: 28.505640
(Iteration 431 / 10000) loss: 28.171576
(Iteration 441 / 10000) loss: 25.515357
(Iteration 451 / 10000) loss: 24.055718
(Iteration 461 / 10000) loss: 28.984333
(Iteration 471 / 10000) loss: 28.827550
(Iteration 481 / 10000) loss: 24.218338
(Iteration 491 / 10000) loss: 28.347303
(Iteration 501 / 10000) loss: 27.622751
(Iteration 511 / 10000) loss: 23.939057
(Iteration 521 / 10000) loss: 26.755916
(Iteration 531 / 10000) loss: 26.010806
(Iteration 541 / 10000) loss: 25.927289
(Iteration 551 / 10000) loss: 23.577932
(Iteration 561 / 10000) loss: 21.839295
(Iteration 571 / 10000) loss: 23.433892
```

```
(Iteration 581 / 10000) loss: 25.823161
(Iteration 591 / 10000) loss: 24.499018
(Iteration 601 / 10000) loss: 23.221837
(Iteration 611 / 10000) loss: 25.673646
(Iteration 621 / 10000) loss: 23.509110
(Iteration 631 / 10000) loss: 24.492510
(Iteration 641 / 10000) loss: 21.759781
(Iteration 651 / 10000) loss: 24.802936
(Iteration 661 / 10000) loss: 23.329650
(Iteration 671 / 10000) loss: 22.937408
(Iteration 681 / 10000) loss: 21.683627
(Iteration 691 / 10000) loss: 23.303339
(Iteration 701 / 10000) loss: 22.999120
(Iteration 711 / 10000) loss: 24.107198
(Iteration 721 / 10000) loss: 20.989781
(Iteration 731 / 10000) loss: 20.452402
(Iteration 741 / 10000) loss: 26.845369
(Iteration 751 / 10000) loss: 20.544558
(Iteration 761 / 10000) loss: 16.946730
(Iteration 771 / 10000) loss: 27.330660
(Iteration 781 / 10000) loss: 25.168350
(Iteration 791 / 10000) loss: 19.014944
(Iteration 801 / 10000) loss: 22.538804
(Iteration 811 / 10000) loss: 21.706520
(Iteration 821 / 10000) loss: 21.565940
(Iteration 831 / 10000) loss: 23.010879
(Iteration 841 / 10000) loss: 24.544214
(Iteration 851 / 10000) loss: 22.670332
(Iteration 861 / 10000) loss: 21.910099
(Iteration 871 / 10000) loss: 24.049240
(Iteration 881 / 10000) loss: 22.290084
(Iteration 891 / 10000) loss: 18.767493
(Iteration 901 / 10000) loss: 23.324185
(Iteration 911 / 10000) loss: 23.151323
(Iteration 921 / 10000) loss: 22.348181
(Iteration 931 / 10000) loss: 18.084972
(Iteration 941 / 10000) loss: 17.947874
(Iteration 951 / 10000) loss: 21.434827
(Iteration 961 / 10000) loss: 17.541884
(Iteration 971 / 10000) loss: 21.517197
(Iteration 981 / 10000) loss: 22.726007
(Iteration 991 / 10000) loss: 20.007530
(Iteration 1001 / 10000) loss: 16.520203
(Iteration 1011 / 10000) loss: 21.046971
(Iteration 1021 / 10000) loss: 19.246777
(Iteration 1031 / 10000) loss: 20.182605
(Iteration 1041 / 10000) loss: 21.487890
(Iteration 1051 / 10000) loss: 21.056914
```

```
(Iteration 1061 / 10000) loss: 19.535169
(Iteration 1071 / 10000) loss: 18.715388
(Iteration 1081 / 10000) loss: 16.141629
(Iteration 1091 / 10000) loss: 22.317019
(Iteration 1101 / 10000) loss: 25.274963
(Iteration 1111 / 10000) loss: 19.796291
(Iteration 1121 / 10000) loss: 18.706015
(Iteration 1131 / 10000) loss: 20.668363
(Iteration 1141 / 10000) loss: 18.860805
(Iteration 1151 / 10000) loss: 17.566043
(Iteration 1161 / 10000) loss: 17.222966
(Iteration 1171 / 10000) loss: 17.843923
(Iteration 1181 / 10000) loss: 17.179211
(Iteration 1191 / 10000) loss: 18.550746
(Iteration 1201 / 10000) loss: 18.573764
(Iteration 1211 / 10000) loss: 16.494365
(Iteration 1221 / 10000) loss: 20.207332
(Iteration 1231 / 10000) loss: 15.895357
(Iteration 1241 / 10000) loss: 14.432801
(Iteration 1251 / 10000) loss: 16.454549
(Iteration 1261 / 10000) loss: 21.223386
(Iteration 1271 / 10000) loss: 18.000194
(Iteration 1281 / 10000) loss: 15.732315
(Iteration 1291 / 10000) loss: 16.728246
(Iteration 1301 / 10000) loss: 20.770832
(Iteration 1311 / 10000) loss: 16.930829
(Iteration 1321 / 10000) loss: 17.053908
(Iteration 1331 / 10000) loss: 17.750995
(Iteration 1341 / 10000) loss: 21.790971
(Iteration 1351 / 10000) loss: 18.312846
(Iteration 1361 / 10000) loss: 16.646212
(Iteration 1371 / 10000) loss: 15.250757
(Iteration 1381 / 10000) loss: 15.115761
(Iteration 1391 / 10000) loss: 16.236158
(Iteration 1401 / 10000) loss: 14.456373
(Iteration 1411 / 10000) loss: 14.400279
(Iteration 1421 / 10000) loss: 19.195996
(Iteration 1431 / 10000) loss: 14.579241
(Iteration 1441 / 10000) loss: 14.826941
(Iteration 1451 / 10000) loss: 17.265090
(Iteration 1461 / 10000) loss: 17.204246
(Iteration 1471 / 10000) loss: 16.788892
(Iteration 1481 / 10000) loss: 15.728334
(Iteration 1491 / 10000) loss: 15.331083
(Iteration 1501 / 10000) loss: 16.724106
(Iteration 1511 / 10000) loss: 15.651230
(Iteration 1521 / 10000) loss: 17.017758
(Iteration 1531 / 10000) loss: 18.119486
```

```
(Iteration 1541 / 10000) loss: 20.825503
(Iteration 1551 / 10000) loss: 16.166433
(Iteration 1561 / 10000) loss: 18.770026
(Iteration 1571 / 10000) loss: 13.137452
(Iteration 1581 / 10000) loss: 14.458733
(Iteration 1591 / 10000) loss: 15.417127
(Iteration 1601 / 10000) loss: 16.014134
(Iteration 1611 / 10000) loss: 15.459367
(Iteration 1621 / 10000) loss: 16.642802
(Iteration 1631 / 10000) loss: 14.986944
(Iteration 1641 / 10000) loss: 16.128539
(Iteration 1651 / 10000) loss: 15.705505
(Iteration 1661 / 10000) loss: 15.890609
(Iteration 1671 / 10000) loss: 16.500009
(Iteration 1681 / 10000) loss: 13.021557
(Iteration 1691 / 10000) loss: 17.245289
(Iteration 1701 / 10000) loss: 16.224758
(Iteration 1711 / 10000) loss: 16.297354
(Iteration 1721 / 10000) loss: 18.979173
(Iteration 1731 / 10000) loss: 16.252367
(Iteration 1741 / 10000) loss: 14.283140
(Iteration 1751 / 10000) loss: 14.348764
(Iteration 1761 / 10000) loss: 14.403134
(Iteration 1771 / 10000) loss: 14.828319
(Iteration 1781 / 10000) loss: 14.733056
(Iteration 1791 / 10000) loss: 14.731680
(Iteration 1801 / 10000) loss: 13.003896
(Iteration 1811 / 10000) loss: 14.151400
(Iteration 1821 / 10000) loss: 16.211521
(Iteration 1831 / 10000) loss: 13.823058
(Iteration 1841 / 10000) loss: 15.035237
(Iteration 1851 / 10000) loss: 13.907551
(Iteration 1861 / 10000) loss: 14.435944
(Iteration 1871 / 10000) loss: 14.182080
(Iteration 1881 / 10000) loss: 14.291607
(Iteration 1891 / 10000) loss: 14.563774
(Iteration 1901 / 10000) loss: 17.885469
(Iteration 1911 / 10000) loss: 16.706508
(Iteration 1921 / 10000) loss: 12.665744
(Iteration 1931 / 10000) loss: 11.472915
(Iteration 1941 / 10000) loss: 13.542350
(Iteration 1951 / 10000) loss: 15.428391
(Iteration 1961 / 10000) loss: 12.412390
(Iteration 1971 / 10000) loss: 14.856977
(Iteration 1981 / 10000) loss: 14.382290
(Iteration 1991 / 10000) loss: 13.332196
(Iteration 2001 / 10000) loss: 17.678868
(Iteration 2011 / 10000) loss: 11.807108
```

```
(Iteration 2021 / 10000) loss: 13.529039
(Iteration 2031 / 10000) loss: 13.957234
(Iteration 2041 / 10000) loss: 13.343297
(Iteration 2051 / 10000) loss: 12.500120
(Iteration 2061 / 10000) loss: 15.201938
(Iteration 2071 / 10000) loss: 14.001454
(Iteration 2081 / 10000) loss: 11.888307
(Iteration 2091 / 10000) loss: 14.437073
(Iteration 2101 / 10000) loss: 13.027301
(Iteration 2111 / 10000) loss: 11.784046
(Iteration 2121 / 10000) loss: 12.860114
(Iteration 2131 / 10000) loss: 13.288853
(Iteration 2141 / 10000) loss: 11.699139
(Iteration 2151 / 10000) loss: 13.940073
(Iteration 2161 / 10000) loss: 11.738961
(Iteration 2171 / 10000) loss: 12.108687
(Iteration 2181 / 10000) loss: 10.888705
(Iteration 2191 / 10000) loss: 14.380386
(Iteration 2201 / 10000) loss: 12.989289
(Iteration 2211 / 10000) loss: 13.268139
(Iteration 2221 / 10000) loss: 14.310404
(Iteration 2231 / 10000) loss: 14.869535
(Iteration 2241 / 10000) loss: 14.320727
(Iteration 2251 / 10000) loss: 14.537993
(Iteration 2261 / 10000) loss: 9.844080
(Iteration 2271 / 10000) loss: 11.970950
(Iteration 2281 / 10000) loss: 11.488095
(Iteration 2291 / 10000) loss: 13.307488
(Iteration 2301 / 10000) loss: 10.869630
(Iteration 2311 / 10000) loss: 13.917210
(Iteration 2321 / 10000) loss: 14.312722
(Iteration 2331 / 10000) loss: 18.225878
(Iteration 2341 / 10000) loss: 16.269078
(Iteration 2351 / 10000) loss: 15.683230
(Iteration 2361 / 10000) loss: 11.218254
(Iteration 2371 / 10000) loss: 13.559671
(Iteration 2381 / 10000) loss: 14.780902
(Iteration 2391 / 10000) loss: 12.426844
(Iteration 2401 / 10000) loss: 15.221593
(Iteration 2411 / 10000) loss: 12.859445
(Iteration 2421 / 10000) loss: 12.436610
(Iteration 2431 / 10000) loss: 14.359621
(Iteration 2441 / 10000) loss: 14.263073
(Iteration 2451 / 10000) loss: 14.326717
(Iteration 2461 / 10000) loss: 12.056367
(Iteration 2471 / 10000) loss: 14.856183
(Iteration 2481 / 10000) loss: 12.925509
(Iteration 2491 / 10000) loss: 13.681111
```

```
(Iteration 2501 / 10000) loss: 15.388741
(Iteration 2511 / 10000) loss: 14.233107
(Iteration 2521 / 10000) loss: 15.662368
(Iteration 2531 / 10000) loss: 11.689503
(Iteration 2541 / 10000) loss: 10.551997
(Iteration 2551 / 10000) loss: 11.269098
(Iteration 2561 / 10000) loss: 15.278555
(Iteration 2571 / 10000) loss: 10.921681
(Iteration 2581 / 10000) loss: 14.242297
(Iteration 2591 / 10000) loss: 11.953889
(Iteration 2601 / 10000) loss: 14.047072
(Iteration 2611 / 10000) loss: 13.539825
(Iteration 2621 / 10000) loss: 10.978116
(Iteration 2631 / 10000) loss: 12.035687
(Iteration 2641 / 10000) loss: 11.651895
(Iteration 2651 / 10000) loss: 12.233355
(Iteration 2661 / 10000) loss: 12.902073
(Iteration 2671 / 10000) loss: 14.112139
(Iteration 2681 / 10000) loss: 12.208501
(Iteration 2691 / 10000) loss: 12.860892
(Iteration 2701 / 10000) loss: 13.159910
(Iteration 2711 / 10000) loss: 12.509091
(Iteration 2721 / 10000) loss: 11.893936
(Iteration 2731 / 10000) loss: 11.985028
(Iteration 2741 / 10000) loss: 9.502390
(Iteration 2751 / 10000) loss: 12.020822
(Iteration 2761 / 10000) loss: 12.234993
(Iteration 2771 / 10000) loss: 12.676166
(Iteration 2781 / 10000) loss: 9.959099
(Iteration 2791 / 10000) loss: 11.070548
(Iteration 2801 / 10000) loss: 13.986011
(Iteration 2811 / 10000) loss: 12.650733
(Iteration 2821 / 10000) loss: 13.804542
(Iteration 2831 / 10000) loss: 11.039493
(Iteration 2841 / 10000) loss: 12.683291
(Iteration 2851 / 10000) loss: 12.440110
(Iteration 2861 / 10000) loss: 13.325308
(Iteration 2871 / 10000) loss: 11.124426
(Iteration 2881 / 10000) loss: 11.269694
(Iteration 2891 / 10000) loss: 11.578021
(Iteration 2901 / 10000) loss: 13.257281
(Iteration 2911 / 10000) loss: 10.235547
(Iteration 2921 / 10000) loss: 13.801733
(Iteration 2931 / 10000) loss: 13.510471
(Iteration 2941 / 10000) loss: 12.142390
(Iteration 2951 / 10000) loss: 11.274170
(Iteration 2961 / 10000) loss: 12.807893
(Iteration 2971 / 10000) loss: 12.388918
```

```
(Iteration 2981 / 10000) loss: 8.905060
(Iteration 2991 / 10000) loss: 11.498371
(Iteration 3001 / 10000) loss: 13.585428
(Iteration 3011 / 10000) loss: 15.260557
(Iteration 3021 / 10000) loss: 11.549980
(Iteration 3031 / 10000) loss: 10.499758
(Iteration 3041 / 10000) loss: 9.786125
(Iteration 3051 / 10000) loss: 12.849122
(Iteration 3061 / 10000) loss: 10.861226
(Iteration 3071 / 10000) loss: 10.897287
(Iteration 3081 / 10000) loss: 10.683367
(Iteration 3091 / 10000) loss: 14.419148
(Iteration 3101 / 10000) loss: 10.826059
(Iteration 3111 / 10000) loss: 12.637522
(Iteration 3121 / 10000) loss: 13.299114
(Iteration 3131 / 10000) loss: 10.926489
(Iteration 3141 / 10000) loss: 13.827043
(Iteration 3151 / 10000) loss: 10.894008
(Iteration 3161 / 10000) loss: 12.222799
(Iteration 3171 / 10000) loss: 8.799823
(Iteration 3181 / 10000) loss: 12.958483
(Iteration 3191 / 10000) loss: 11.065429
(Iteration 3201 / 10000) loss: 14.510665
(Iteration 3211 / 10000) loss: 10.533061
(Iteration 3221 / 10000) loss: 13.102975
(Iteration 3231 / 10000) loss: 12.410626
(Iteration 3241 / 10000) loss: 10.591996
(Iteration 3251 / 10000) loss: 9.446178
(Iteration 3261 / 10000) loss: 10.344863
(Iteration 3271 / 10000) loss: 14.938346
(Iteration 3281 / 10000) loss: 10.994919
(Iteration 3291 / 10000) loss: 12.529604
(Iteration 3301 / 10000) loss: 11.400762
(Iteration 3311 / 10000) loss: 14.761403
(Iteration 3321 / 10000) loss: 8.902695
(Iteration 3331 / 10000) loss: 10.520322
(Iteration 3341 / 10000) loss: 13.896164
(Iteration 3351 / 10000) loss: 11.192266
(Iteration 3361 / 10000) loss: 11.086080
(Iteration 3371 / 10000) loss: 10.933654
(Iteration 3381 / 10000) loss: 11.355337
(Iteration 3391 / 10000) loss: 11.187919
(Iteration 3401 / 10000) loss: 8.329341
(Iteration 3411 / 10000) loss: 11.051116
(Iteration 3421 / 10000) loss: 9.640642
(Iteration 3431 / 10000) loss: 12.190350
(Iteration 3441 / 10000) loss: 10.315090
(Iteration 3451 / 10000) loss: 11.300133
```

```
(Iteration 3461 / 10000) loss: 10.147561
(Iteration 3471 / 10000) loss: 10.114984
(Iteration 3481 / 10000) loss: 12.169260
(Iteration 3491 / 10000) loss: 10.989205
(Iteration 3501 / 10000) loss: 10.123063
(Iteration 3511 / 10000) loss: 13.221140
(Iteration 3521 / 10000) loss: 9.846463
(Iteration 3531 / 10000) loss: 10.423045
(Iteration 3541 / 10000) loss: 10.990733
(Iteration 3551 / 10000) loss: 10.387945
(Iteration 3561 / 10000) loss: 10.400827
(Iteration 3571 / 10000) loss: 10.213137
(Iteration 3581 / 10000) loss: 11.165744
(Iteration 3591 / 10000) loss: 9.883789
(Iteration 3601 / 10000) loss: 10.133688
(Iteration 3611 / 10000) loss: 11.260068
(Iteration 3621 / 10000) loss: 9.818782
(Iteration 3631 / 10000) loss: 11.143315
(Iteration 3641 / 10000) loss: 12.578553
(Iteration 3651 / 10000) loss: 9.077655
(Iteration 3661 / 10000) loss: 12.252743
(Iteration 3671 / 10000) loss: 9.945444
(Iteration 3681 / 10000) loss: 10.776642
(Iteration 3691 / 10000) loss: 10.518280
(Iteration 3701 / 10000) loss: 11.023761
(Iteration 3711 / 10000) loss: 9.438974
(Iteration 3721 / 10000) loss: 10.599352
(Iteration 3731 / 10000) loss: 10.260268
(Iteration 3741 / 10000) loss: 12.141759
(Iteration 3751 / 10000) loss: 10.582303
(Iteration 3761 / 10000) loss: 11.489039
(Iteration 3771 / 10000) loss: 10.619055
(Iteration 3781 / 10000) loss: 10.813915
(Iteration 3791 / 10000) loss: 10.596387
(Iteration 3801 / 10000) loss: 8.559876
(Iteration 3811 / 10000) loss: 12.164913
(Iteration 3821 / 10000) loss: 10.552043
(Iteration 3831 / 10000) loss: 10.395254
(Iteration 3841 / 10000) loss: 11.080640
(Iteration 3851 / 10000) loss: 13.909364
(Iteration 3861 / 10000) loss: 12.248432
(Iteration 3871 / 10000) loss: 9.704909
(Iteration 3881 / 10000) loss: 10.949049
(Iteration 3891 / 10000) loss: 9.684814
(Iteration 3901 / 10000) loss: 9.391348
(Iteration 3911 / 10000) loss: 10.677471
(Iteration 3921 / 10000) loss: 11.254231
(Iteration 3931 / 10000) loss: 12.900263
```

```
(Iteration 3941 / 10000) loss: 9.272728
(Iteration 3951 / 10000) loss: 10.226641
(Iteration 3961 / 10000) loss: 10.167268
(Iteration 3971 / 10000) loss: 12.832012
(Iteration 3981 / 10000) loss: 8.892122
(Iteration 3991 / 10000) loss: 11.577384
(Iteration 4001 / 10000) loss: 8.694315
(Iteration 4011 / 10000) loss: 12.095782
(Iteration 4021 / 10000) loss: 9.571767
(Iteration 4031 / 10000) loss: 10.517373
(Iteration 4041 / 10000) loss: 11.450652
(Iteration 4051 / 10000) loss: 10.440612
(Iteration 4061 / 10000) loss: 11.203720
(Iteration 4071 / 10000) loss: 11.932363
(Iteration 4081 / 10000) loss: 10.139232
(Iteration 4091 / 10000) loss: 10.294964
(Iteration 4101 / 10000) loss: 13.091309
(Iteration 4111 / 10000) loss: 8.580089
(Iteration 4121 / 10000) loss: 12.356384
(Iteration 4131 / 10000) loss: 10.758738
(Iteration 4141 / 10000) loss: 9.893984
(Iteration 4151 / 10000) loss: 12.156481
(Iteration 4161 / 10000) loss: 11.880679
(Iteration 4171 / 10000) loss: 11.028222
(Iteration 4181 / 10000) loss: 10.152898
(Iteration 4191 / 10000) loss: 10.852725
(Iteration 4201 / 10000) loss: 9.843966
(Iteration 4211 / 10000) loss: 10.000867
(Iteration 4221 / 10000) loss: 9.726166
(Iteration 4231 / 10000) loss: 12.503579
(Iteration 4241 / 10000) loss: 10.588968
(Iteration 4251 / 10000) loss: 10.012796
(Iteration 4261 / 10000) loss: 11.473891
(Iteration 4271 / 10000) loss: 11.009241
(Iteration 4281 / 10000) loss: 12.293768
(Iteration 4291 / 10000) loss: 10.974051
(Iteration 4301 / 10000) loss: 11.193461
(Iteration 4311 / 10000) loss: 9.589305
(Iteration 4321 / 10000) loss: 10.265208
(Iteration 4331 / 10000) loss: 11.074107
(Iteration 4341 / 10000) loss: 10.659160
(Iteration 4351 / 10000) loss: 10.468486
(Iteration 4361 / 10000) loss: 11.382752
(Iteration 4371 / 10000) loss: 10.153065
(Iteration 4381 / 10000) loss: 9.969725
(Iteration 4391 / 10000) loss: 10.284709
(Iteration 4401 / 10000) loss: 12.162268
(Iteration 4411 / 10000) loss: 11.068002
```

```
(Iteration 4421 / 10000) loss: 10.212407
(Iteration 4431 / 10000) loss: 6.478555
(Iteration 4441 / 10000) loss: 9.879390
(Iteration 4451 / 10000) loss: 10.847990
(Iteration 4461 / 10000) loss: 11.604446
(Iteration 4471 / 10000) loss: 8.513664
(Iteration 4481 / 10000) loss: 10.279165
(Iteration 4491 / 10000) loss: 11.926483
(Iteration 4501 / 10000) loss: 8.989251
(Iteration 4511 / 10000) loss: 9.299380
(Iteration 4521 / 10000) loss: 8.787059
(Iteration 4531 / 10000) loss: 8.341896
(Iteration 4541 / 10000) loss: 7.982159
(Iteration 4551 / 10000) loss: 10.283215
(Iteration 4561 / 10000) loss: 10.533693
(Iteration 4571 / 10000) loss: 10.484091
(Iteration 4581 / 10000) loss: 9.482438
(Iteration 4591 / 10000) loss: 7.927515
(Iteration 4601 / 10000) loss: 10.843690
(Iteration 4611 / 10000) loss: 10.433941
(Iteration 4621 / 10000) loss: 8.103319
(Iteration 4631 / 10000) loss: 9.669329
(Iteration 4641 / 10000) loss: 11.811203
(Iteration 4651 / 10000) loss: 9.157195
(Iteration 4661 / 10000) loss: 8.708216
(Iteration 4671 / 10000) loss: 10.419049
(Iteration 4681 / 10000) loss: 9.899025
(Iteration 4691 / 10000) loss: 10.939032
(Iteration 4701 / 10000) loss: 7.815035
(Iteration 4711 / 10000) loss: 12.073573
(Iteration 4721 / 10000) loss: 9.137045
(Iteration 4731 / 10000) loss: 10.322579
(Iteration 4741 / 10000) loss: 11.880354
(Iteration 4751 / 10000) loss: 11.045758
(Iteration 4761 / 10000) loss: 10.455342
(Iteration 4771 / 10000) loss: 10.954025
(Iteration 4781 / 10000) loss: 9.716953
(Iteration 4791 / 10000) loss: 11.746325
(Iteration 4801 / 10000) loss: 10.708504
(Iteration 4811 / 10000) loss: 9.222204
(Iteration 4821 / 10000) loss: 10.994254
(Iteration 4831 / 10000) loss: 10.121216
(Iteration 4841 / 10000) loss: 8.713364
(Iteration 4851 / 10000) loss: 10.556440
(Iteration 4861 / 10000) loss: 11.075133
(Iteration 4871 / 10000) loss: 10.068094
(Iteration 4881 / 10000) loss: 9.867685
(Iteration 4891 / 10000) loss: 10.564567
```

```
(Iteration 4901 / 10000) loss: 11.145484
(Iteration 4911 / 10000) loss: 10.902013
(Iteration 4921 / 10000) loss: 9.409594
(Iteration 4931 / 10000) loss: 8.925547
(Iteration 4941 / 10000) loss: 10.346515
(Iteration 4951 / 10000) loss: 11.382185
(Iteration 4961 / 10000) loss: 8.118590
(Iteration 4971 / 10000) loss: 9.047606
(Iteration 4981 / 10000) loss: 7.281624
(Iteration 4991 / 10000) loss: 9.322049
(Iteration 5001 / 10000) loss: 9.193391
(Iteration 5011 / 10000) loss: 10.095583
(Iteration 5021 / 10000) loss: 7.838727
(Iteration 5031 / 10000) loss: 10.934419
(Iteration 5041 / 10000) loss: 10.898797
(Iteration 5051 / 10000) loss: 10.847457
(Iteration 5061 / 10000) loss: 10.793418
(Iteration 5071 / 10000) loss: 8.356976
(Iteration 5081 / 10000) loss: 12.040467
(Iteration 5091 / 10000) loss: 10.060729
(Iteration 5101 / 10000) loss: 8.168450
(Iteration 5111 / 10000) loss: 9.336179
(Iteration 5121 / 10000) loss: 10.120269
(Iteration 5131 / 10000) loss: 10.177018
(Iteration 5141 / 10000) loss: 10.372734
(Iteration 5151 / 10000) loss: 9.984574
(Iteration 5161 / 10000) loss: 10.938375
(Iteration 5171 / 10000) loss: 9.393487
(Iteration 5181 / 10000) loss: 8.440401
(Iteration 5191 / 10000) loss: 11.335632
(Iteration 5201 / 10000) loss: 9.085556
(Iteration 5211 / 10000) loss: 8.849247
(Iteration 5221 / 10000) loss: 10.235788
(Iteration 5231 / 10000) loss: 10.005839
(Iteration 5241 / 10000) loss: 10.985662
(Iteration 5251 / 10000) loss: 9.942590
(Iteration 5261 / 10000) loss: 9.599866
(Iteration 5271 / 10000) loss: 11.972419
(Iteration 5281 / 10000) loss: 9.109536
(Iteration 5291 / 10000) loss: 8.604838
(Iteration 5301 / 10000) loss: 9.605723
(Iteration 5311 / 10000) loss: 9.170627
(Iteration 5321 / 10000) loss: 9.812761
(Iteration 5331 / 10000) loss: 10.322435
(Iteration 5341 / 10000) loss: 8.216701
(Iteration 5351 / 10000) loss: 7.891389
(Iteration 5361 / 10000) loss: 10.804259
(Iteration 5371 / 10000) loss: 8.795367
```

```
(Iteration 5381 / 10000) loss: 9.052739
(Iteration 5391 / 10000) loss: 8.712952
(Iteration 5401 / 10000) loss: 9.397201
(Iteration 5411 / 10000) loss: 9.905018
(Iteration 5421 / 10000) loss: 7.765821
(Iteration 5431 / 10000) loss: 9.872533
(Iteration 5441 / 10000) loss: 8.388774
(Iteration 5451 / 10000) loss: 7.896369
(Iteration 5461 / 10000) loss: 10.393787
(Iteration 5471 / 10000) loss: 8.624268
(Iteration 5481 / 10000) loss: 10.162984
(Iteration 5491 / 10000) loss: 10.499359
(Iteration 5501 / 10000) loss: 8.227524
(Iteration 5511 / 10000) loss: 10.169624
(Iteration 5521 / 10000) loss: 9.401410
(Iteration 5531 / 10000) loss: 9.723614
(Iteration 5541 / 10000) loss: 9.568107
(Iteration 5551 / 10000) loss: 9.349947
(Iteration 5561 / 10000) loss: 10.646604
(Iteration 5571 / 10000) loss: 10.466114
(Iteration 5581 / 10000) loss: 10.701994
(Iteration 5591 / 10000) loss: 9.155265
(Iteration 5601 / 10000) loss: 10.052237
(Iteration 5611 / 10000) loss: 8.485187
(Iteration 5621 / 10000) loss: 8.886542
(Iteration 5631 / 10000) loss: 11.388413
(Iteration 5641 / 10000) loss: 8.987962
(Iteration 5651 / 10000) loss: 9.557634
(Iteration 5661 / 10000) loss: 10.069269
(Iteration 5671 / 10000) loss: 11.379377
(Iteration 5681 / 10000) loss: 10.083783
(Iteration 5691 / 10000) loss: 10.051165
(Iteration 5701 / 10000) loss: 8.255706
(Iteration 5711 / 10000) loss: 10.315731
(Iteration 5721 / 10000) loss: 8.241582
(Iteration 5731 / 10000) loss: 9.177030
(Iteration 5741 / 10000) loss: 12.159460
(Iteration 5751 / 10000) loss: 7.919114
(Iteration 5761 / 10000) loss: 10.412105
(Iteration 5771 / 10000) loss: 10.721675
(Iteration 5781 / 10000) loss: 7.898594
(Iteration 5791 / 10000) loss: 8.940096
(Iteration 5801 / 10000) loss: 10.029937
(Iteration 5811 / 10000) loss: 10.435910
(Iteration 5821 / 10000) loss: 11.982545
(Iteration 5831 / 10000) loss: 8.755623
(Iteration 5841 / 10000) loss: 10.155469
(Iteration 5851 / 10000) loss: 10.066795
```

```
(Iteration 5861 / 10000) loss: 9.392605
(Iteration 5871 / 10000) loss: 7.834576
(Iteration 5881 / 10000) loss: 9.042994
(Iteration 5891 / 10000) loss: 7.970233
(Iteration 5901 / 10000) loss: 8.624903
(Iteration 5911 / 10000) loss: 9.803861
(Iteration 5921 / 10000) loss: 9.652198
(Iteration 5931 / 10000) loss: 8.744590
(Iteration 5941 / 10000) loss: 10.571035
(Iteration 5951 / 10000) loss: 9.608184
(Iteration 5961 / 10000) loss: 8.193167
(Iteration 5971 / 10000) loss: 7.658905
(Iteration 5981 / 10000) loss: 9.516635
(Iteration 5991 / 10000) loss: 9.303897
(Iteration 6001 / 10000) loss: 10.522742
(Iteration 6011 / 10000) loss: 8.630878
(Iteration 6021 / 10000) loss: 10.228420
(Iteration 6031 / 10000) loss: 10.396439
(Iteration 6041 / 10000) loss: 9.396551
(Iteration 6051 / 10000) loss: 9.598820
(Iteration 6061 / 10000) loss: 8.729046
(Iteration 6071 / 10000) loss: 7.535593
(Iteration 6081 / 10000) loss: 8.977728
(Iteration 6091 / 10000) loss: 12.229772
(Iteration 6101 / 10000) loss: 8.369324
(Iteration 6111 / 10000) loss: 9.103518
(Iteration 6121 / 10000) loss: 8.535738
(Iteration 6131 / 10000) loss: 9.323736
(Iteration 6141 / 10000) loss: 9.128605
(Iteration 6151 / 10000) loss: 10.896219
(Iteration 6161 / 10000) loss: 8.926656
(Iteration 6171 / 10000) loss: 10.638094
(Iteration 6181 / 10000) loss: 7.508967
(Iteration 6191 / 10000) loss: 9.416846
(Iteration 6201 / 10000) loss: 9.551871
(Iteration 6211 / 10000) loss: 8.778704
(Iteration 6221 / 10000) loss: 9.202402
(Iteration 6231 / 10000) loss: 8.149878
(Iteration 6241 / 10000) loss: 8.740277
(Iteration 6251 / 10000) loss: 8.817457
(Iteration 6261 / 10000) loss: 7.957153
(Iteration 6271 / 10000) loss: 8.666839
(Iteration 6281 / 10000) loss: 7.521348
(Iteration 6291 / 10000) loss: 8.790672
(Iteration 6301 / 10000) loss: 10.411659
(Iteration 6311 / 10000) loss: 10.227004
(Iteration 6321 / 10000) loss: 8.023343
(Iteration 6331 / 10000) loss: 9.893532
```

```
(Iteration 6341 / 10000) loss: 8.542298
(Iteration 6351 / 10000) loss: 10.671005
(Iteration 6361 / 10000) loss: 8.164044
(Iteration 6371 / 10000) loss: 9.460649
(Iteration 6381 / 10000) loss: 10.633298
(Iteration 6391 / 10000) loss: 6.988837
(Iteration 6401 / 10000) loss: 8.436755
(Iteration 6411 / 10000) loss: 10.173938
(Iteration 6421 / 10000) loss: 7.397735
(Iteration 6431 / 10000) loss: 8.070028
(Iteration 6441 / 10000) loss: 8.551802
(Iteration 6451 / 10000) loss: 7.958542
(Iteration 6461 / 10000) loss: 7.731813
(Iteration 6471 / 10000) loss: 8.855070
(Iteration 6481 / 10000) loss: 7.525476
(Iteration 6491 / 10000) loss: 7.234085
(Iteration 6501 / 10000) loss: 8.197308
(Iteration 6511 / 10000) loss: 9.340612
(Iteration 6521 / 10000) loss: 10.409861
(Iteration 6531 / 10000) loss: 8.438504
(Iteration 6541 / 10000) loss: 11.430245
(Iteration 6551 / 10000) loss: 10.626494
(Iteration 6561 / 10000) loss: 10.058484
(Iteration 6571 / 10000) loss: 9.923206
(Iteration 6581 / 10000) loss: 7.253703
(Iteration 6591 / 10000) loss: 8.008775
(Iteration 6601 / 10000) loss: 9.180560
(Iteration 6611 / 10000) loss: 8.860938
(Iteration 6621 / 10000) loss: 9.216806
(Iteration 6631 / 10000) loss: 8.616625
(Iteration 6641 / 10000) loss: 7.934611
(Iteration 6651 / 10000) loss: 8.187164
(Iteration 6661 / 10000) loss: 9.623804
(Iteration 6671 / 10000) loss: 9.232403
(Iteration 6681 / 10000) loss: 9.950752
(Iteration 6691 / 10000) loss: 10.011723
(Iteration 6701 / 10000) loss: 7.951251
(Iteration 6711 / 10000) loss: 8.526309
(Iteration 6721 / 10000) loss: 9.893980
(Iteration 6731 / 10000) loss: 7.761685
(Iteration 6741 / 10000) loss: 10.002168
(Iteration 6751 / 10000) loss: 8.347420
(Iteration 6761 / 10000) loss: 9.227496
(Iteration 6771 / 10000) loss: 12.659581
(Iteration 6781 / 10000) loss: 10.341212
(Iteration 6791 / 10000) loss: 9.176514
(Iteration 6801 / 10000) loss: 8.847740
(Iteration 6811 / 10000) loss: 7.780979
```

```
(Iteration 6821 / 10000) loss: 8.333584
(Iteration 6831 / 10000) loss: 8.465843
(Iteration 6841 / 10000) loss: 9.183000
(Iteration 6851 / 10000) loss: 9.247467
(Iteration 6861 / 10000) loss: 9.134356
(Iteration 6871 / 10000) loss: 9.818047
(Iteration 6881 / 10000) loss: 8.177580
(Iteration 6891 / 10000) loss: 10.714286
(Iteration 6901 / 10000) loss: 7.033089
(Iteration 6911 / 10000) loss: 9.344031
(Iteration 6921 / 10000) loss: 7.439171
(Iteration 6931 / 10000) loss: 10.664635
(Iteration 6941 / 10000) loss: 6.908500
(Iteration 6951 / 10000) loss: 7.528387
(Iteration 6961 / 10000) loss: 8.096427
(Iteration 6971 / 10000) loss: 10.480122
(Iteration 6981 / 10000) loss: 8.574250
(Iteration 6991 / 10000) loss: 9.533892
(Iteration 7001 / 10000) loss: 8.979835
(Iteration 7011 / 10000) loss: 8.384961
(Iteration 7021 / 10000) loss: 9.047613
(Iteration 7031 / 10000) loss: 6.886488
(Iteration 7041 / 10000) loss: 7.704819
(Iteration 7051 / 10000) loss: 9.424108
(Iteration 7061 / 10000) loss: 9.273762
(Iteration 7071 / 10000) loss: 12.290665
(Iteration 7081 / 10000) loss: 9.291635
(Iteration 7091 / 10000) loss: 8.543821
(Iteration 7101 / 10000) loss: 9.399134
(Iteration 7111 / 10000) loss: 8.598225
(Iteration 7121 / 10000) loss: 7.903965
(Iteration 7131 / 10000) loss: 8.939145
(Iteration 7141 / 10000) loss: 7.938555
(Iteration 7151 / 10000) loss: 8.429524
(Iteration 7161 / 10000) loss: 7.505687
(Iteration 7171 / 10000) loss: 7.691828
(Iteration 7181 / 10000) loss: 8.952861
(Iteration 7191 / 10000) loss: 8.827841
(Iteration 7201 / 10000) loss: 7.023249
(Iteration 7211 / 10000) loss: 9.278906
(Iteration 7221 / 10000) loss: 8.133027
(Iteration 7231 / 10000) loss: 7.936395
(Iteration 7241 / 10000) loss: 10.099561
(Iteration 7251 / 10000) loss: 9.323999
(Iteration 7261 / 10000) loss: 8.267336
(Iteration 7271 / 10000) loss: 8.986244
(Iteration 7281 / 10000) loss: 7.489070
(Iteration 7291 / 10000) loss: 7.547413
```

```
(Iteration 7301 / 10000) loss: 6.604856
(Iteration 7311 / 10000) loss: 7.974798
(Iteration 7321 / 10000) loss: 7.817058
(Iteration 7331 / 10000) loss: 8.564125
(Iteration 7341 / 10000) loss: 10.385202
(Iteration 7351 / 10000) loss: 8.398616
(Iteration 7361 / 10000) loss: 8.515943
(Iteration 7371 / 10000) loss: 9.422909
(Iteration 7381 / 10000) loss: 7.412469
(Iteration 7391 / 10000) loss: 8.088587
(Iteration 7401 / 10000) loss: 8.567209
(Iteration 7411 / 10000) loss: 9.632122
(Iteration 7421 / 10000) loss: 9.219446
(Iteration 7431 / 10000) loss: 7.367024
(Iteration 7441 / 10000) loss: 8.623553
(Iteration 7451 / 10000) loss: 7.455872
(Iteration 7461 / 10000) loss: 7.791988
(Iteration 7471 / 10000) loss: 8.201691
(Iteration 7481 / 10000) loss: 8.963697
(Iteration 7491 / 10000) loss: 8.155415
(Iteration 7501 / 10000) loss: 8.256746
(Iteration 7511 / 10000) loss: 7.393002
(Iteration 7521 / 10000) loss: 6.798969
(Iteration 7531 / 10000) loss: 9.898127
(Iteration 7541 / 10000) loss: 8.702571
(Iteration 7551 / 10000) loss: 6.535671
(Iteration 7561 / 10000) loss: 7.293694
(Iteration 7571 / 10000) loss: 8.034373
(Iteration 7581 / 10000) loss: 6.864241
(Iteration 7591 / 10000) loss: 7.836465
(Iteration 7601 / 10000) loss: 9.058802
(Iteration 7611 / 10000) loss: 6.495589
(Iteration 7621 / 10000) loss: 7.069921
(Iteration 7631 / 10000) loss: 6.696801
(Iteration 7641 / 10000) loss: 8.763437
(Iteration 7651 / 10000) loss: 8.551381
(Iteration 7661 / 10000) loss: 7.798015
(Iteration 7671 / 10000) loss: 9.094558
(Iteration 7681 / 10000) loss: 9.128492
(Iteration 7691 / 10000) loss: 9.371678
(Iteration 7701 / 10000) loss: 8.462468
(Iteration 7711 / 10000) loss: 7.649551
(Iteration 7721 / 10000) loss: 8.904893
(Iteration 7731 / 10000) loss: 8.724087
(Iteration 7741 / 10000) loss: 7.914146
(Iteration 7751 / 10000) loss: 8.671664
(Iteration 7761 / 10000) loss: 7.620545
(Iteration 7771 / 10000) loss: 9.526067
```

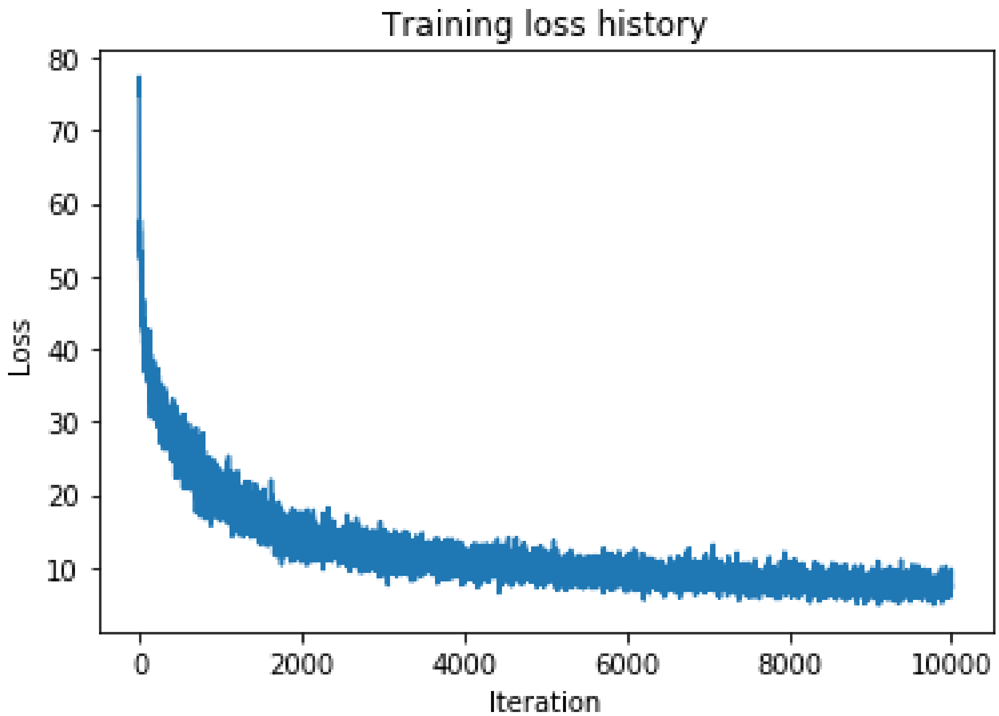
```
(Iteration 7781 / 10000) loss: 8.932662
(Iteration 7791 / 10000) loss: 7.130984
(Iteration 7801 / 10000) loss: 7.904787
(Iteration 7811 / 10000) loss: 7.756123
(Iteration 7821 / 10000) loss: 8.917580
(Iteration 7831 / 10000) loss: 7.732977
(Iteration 7841 / 10000) loss: 9.456774
(Iteration 7851 / 10000) loss: 9.453048
(Iteration 7861 / 10000) loss: 7.365654
(Iteration 7871 / 10000) loss: 8.646851
(Iteration 7881 / 10000) loss: 8.908974
(Iteration 7891 / 10000) loss: 9.609685
(Iteration 7901 / 10000) loss: 7.244018
(Iteration 7911 / 10000) loss: 7.806608
(Iteration 7921 / 10000) loss: 6.906990
(Iteration 7931 / 10000) loss: 8.882545
(Iteration 7941 / 10000) loss: 8.750050
(Iteration 7951 / 10000) loss: 6.495708
(Iteration 7961 / 10000) loss: 8.316061
(Iteration 7971 / 10000) loss: 6.674675
(Iteration 7981 / 10000) loss: 6.576446
(Iteration 7991 / 10000) loss: 10.011291
(Iteration 8001 / 10000) loss: 10.234997
(Iteration 8011 / 10000) loss: 9.326476
(Iteration 8021 / 10000) loss: 8.154320
(Iteration 8031 / 10000) loss: 9.299623
(Iteration 8041 / 10000) loss: 8.386006
(Iteration 8051 / 10000) loss: 9.592711
(Iteration 8061 / 10000) loss: 8.711210
(Iteration 8071 / 10000) loss: 7.297323
(Iteration 8081 / 10000) loss: 8.218335
(Iteration 8091 / 10000) loss: 7.642526
(Iteration 8101 / 10000) loss: 6.764849
(Iteration 8111 / 10000) loss: 9.855438
(Iteration 8121 / 10000) loss: 7.132187
(Iteration 8131 / 10000) loss: 7.199882
(Iteration 8141 / 10000) loss: 6.179191
(Iteration 8151 / 10000) loss: 7.447510
(Iteration 8161 / 10000) loss: 7.772965
(Iteration 8171 / 10000) loss: 7.099766
(Iteration 8181 / 10000) loss: 8.568248
(Iteration 8191 / 10000) loss: 9.404440
(Iteration 8201 / 10000) loss: 9.062830
(Iteration 8211 / 10000) loss: 8.041658
(Iteration 8221 / 10000) loss: 8.055557
(Iteration 8231 / 10000) loss: 7.576169
(Iteration 8241 / 10000) loss: 7.820153
(Iteration 8251 / 10000) loss: 7.728975
```

```
(Iteration 8261 / 10000) loss: 9.295229
(Iteration 8271 / 10000) loss: 8.296870
(Iteration 8281 / 10000) loss: 7.520292
(Iteration 8291 / 10000) loss: 8.451141
(Iteration 8301 / 10000) loss: 7.195191
(Iteration 8311 / 10000) loss: 7.615199
(Iteration 8321 / 10000) loss: 8.855717
(Iteration 8331 / 10000) loss: 10.179992
(Iteration 8341 / 10000) loss: 7.575979
(Iteration 8351 / 10000) loss: 7.896876
(Iteration 8361 / 10000) loss: 7.177388
(Iteration 8371 / 10000) loss: 7.454214
(Iteration 8381 / 10000) loss: 7.444011
(Iteration 8391 / 10000) loss: 9.000510
(Iteration 8401 / 10000) loss: 6.979841
(Iteration 8411 / 10000) loss: 8.727142
(Iteration 8421 / 10000) loss: 8.242331
(Iteration 8431 / 10000) loss: 6.498061
(Iteration 8441 / 10000) loss: 8.871663
(Iteration 8451 / 10000) loss: 8.886436
(Iteration 8461 / 10000) loss: 8.118105
(Iteration 8471 / 10000) loss: 8.650907
(Iteration 8481 / 10000) loss: 7.988346
(Iteration 8491 / 10000) loss: 7.428562
(Iteration 8501 / 10000) loss: 6.761647
(Iteration 8511 / 10000) loss: 7.002001
(Iteration 8521 / 10000) loss: 7.204332
(Iteration 8531 / 10000) loss: 8.298969
(Iteration 8541 / 10000) loss: 7.798928
(Iteration 8551 / 10000) loss: 7.940386
(Iteration 8561 / 10000) loss: 8.101159
(Iteration 8571 / 10000) loss: 7.646421
(Iteration 8581 / 10000) loss: 7.874548
(Iteration 8591 / 10000) loss: 8.163433
(Iteration 8601 / 10000) loss: 7.473157
(Iteration 8611 / 10000) loss: 8.428783
(Iteration 8621 / 10000) loss: 8.287775
(Iteration 8631 / 10000) loss: 7.386813
(Iteration 8641 / 10000) loss: 8.365291
(Iteration 8651 / 10000) loss: 8.200930
(Iteration 8661 / 10000) loss: 8.603658
(Iteration 8671 / 10000) loss: 7.276169
(Iteration 8681 / 10000) loss: 8.441154
(Iteration 8691 / 10000) loss: 7.394439
(Iteration 8701 / 10000) loss: 6.760572
(Iteration 8711 / 10000) loss: 10.353082
(Iteration 8721 / 10000) loss: 7.620495
(Iteration 8731 / 10000) loss: 8.492820
```

```
(Iteration 8741 / 10000) loss: 6.818105
(Iteration 8751 / 10000) loss: 7.364844
(Iteration 8761 / 10000) loss: 7.318272
(Iteration 8771 / 10000) loss: 7.341249
(Iteration 8781 / 10000) loss: 6.748991
(Iteration 8791 / 10000) loss: 7.027331
(Iteration 8801 / 10000) loss: 7.652741
(Iteration 8811 / 10000) loss: 6.572076
(Iteration 8821 / 10000) loss: 5.988093
(Iteration 8831 / 10000) loss: 7.724007
(Iteration 8841 / 10000) loss: 6.570193
(Iteration 8851 / 10000) loss: 6.915420
(Iteration 8861 / 10000) loss: 6.064033
(Iteration 8871 / 10000) loss: 7.701421
(Iteration 8881 / 10000) loss: 9.499383
(Iteration 8891 / 10000) loss: 9.239771
(Iteration 8901 / 10000) loss: 7.058578
(Iteration 8911 / 10000) loss: 7.671011
(Iteration 8921 / 10000) loss: 6.187872
(Iteration 8931 / 10000) loss: 7.792530
(Iteration 8941 / 10000) loss: 7.209829
(Iteration 8951 / 10000) loss: 7.972177
(Iteration 8961 / 10000) loss: 6.278889
(Iteration 8971 / 10000) loss: 8.905732
(Iteration 8981 / 10000) loss: 7.493971
(Iteration 8991 / 10000) loss: 7.862089
(Iteration 9001 / 10000) loss: 7.081863
(Iteration 9011 / 10000) loss: 7.488536
(Iteration 9021 / 10000) loss: 7.578713
(Iteration 9031 / 10000) loss: 7.609693
(Iteration 9041 / 10000) loss: 6.846102
(Iteration 9051 / 10000) loss: 8.183986
(Iteration 9061 / 10000) loss: 7.980236
(Iteration 9071 / 10000) loss: 8.849737
(Iteration 9081 / 10000) loss: 8.719758
(Iteration 9091 / 10000) loss: 9.991550
(Iteration 9101 / 10000) loss: 9.944331
(Iteration 9111 / 10000) loss: 10.672667
(Iteration 9121 / 10000) loss: 6.901811
(Iteration 9131 / 10000) loss: 6.309572
(Iteration 9141 / 10000) loss: 6.195690
(Iteration 9151 / 10000) loss: 7.752680
(Iteration 9161 / 10000) loss: 9.577405
(Iteration 9171 / 10000) loss: 7.300211
(Iteration 9181 / 10000) loss: 7.368996
(Iteration 9191 / 10000) loss: 9.658686
(Iteration 9201 / 10000) loss: 6.935105
(Iteration 9211 / 10000) loss: 6.349467
```

```
(Iteration 9221 / 10000) loss: 7.977553
(Iteration 9231 / 10000) loss: 7.909810
(Iteration 9241 / 10000) loss: 7.313673
(Iteration 9251 / 10000) loss: 7.916474
(Iteration 9261 / 10000) loss: 8.030509
(Iteration 9271 / 10000) loss: 7.640015
(Iteration 9281 / 10000) loss: 5.599151
(Iteration 9291 / 10000) loss: 8.566371
(Iteration 9301 / 10000) loss: 9.097439
(Iteration 9311 / 10000) loss: 5.332582
(Iteration 9321 / 10000) loss: 6.936906
(Iteration 9331 / 10000) loss: 9.666995
(Iteration 9341 / 10000) loss: 7.504153
(Iteration 9351 / 10000) loss: 7.005216
(Iteration 9361 / 10000) loss: 8.446337
(Iteration 9371 / 10000) loss: 8.299245
(Iteration 9381 / 10000) loss: 11.052817
(Iteration 9391 / 10000) loss: 5.491827
(Iteration 9401 / 10000) loss: 6.938379
(Iteration 9411 / 10000) loss: 6.803713
(Iteration 9421 / 10000) loss: 7.146941
(Iteration 9431 / 10000) loss: 6.218255
(Iteration 9441 / 10000) loss: 6.762672
(Iteration 9451 / 10000) loss: 8.035220
(Iteration 9461 / 10000) loss: 7.625431
(Iteration 9471 / 10000) loss: 6.904982
(Iteration 9481 / 10000) loss: 8.793571
(Iteration 9491 / 10000) loss: 7.461753
(Iteration 9501 / 10000) loss: 7.800919
(Iteration 9511 / 10000) loss: 7.740658
(Iteration 9521 / 10000) loss: 6.642001
(Iteration 9531 / 10000) loss: 5.944496
(Iteration 9541 / 10000) loss: 8.327601
(Iteration 9551 / 10000) loss: 7.797910
(Iteration 9561 / 10000) loss: 7.691011
(Iteration 9571 / 10000) loss: 6.799355
(Iteration 9581 / 10000) loss: 7.431069
(Iteration 9591 / 10000) loss: 9.141195
(Iteration 9601 / 10000) loss: 8.065292
(Iteration 9611 / 10000) loss: 8.131534
(Iteration 9621 / 10000) loss: 7.631747
(Iteration 9631 / 10000) loss: 8.323091
(Iteration 9641 / 10000) loss: 5.509583
(Iteration 9651 / 10000) loss: 6.901162
(Iteration 9661 / 10000) loss: 6.844626
(Iteration 9671 / 10000) loss: 9.807609
(Iteration 9681 / 10000) loss: 8.781584
(Iteration 9691 / 10000) loss: 8.622404
```

```
(Iteration 9701 / 10000) loss: 9.356723
(Iteration 9711 / 10000) loss: 8.182191
(Iteration 9721 / 10000) loss: 6.264380
(Iteration 9731 / 10000) loss: 9.442458
(Iteration 9741 / 10000) loss: 7.797177
(Iteration 9751 / 10000) loss: 8.463321
(Iteration 9761 / 10000) loss: 7.886726
(Iteration 9771 / 10000) loss: 6.721421
(Iteration 9781 / 10000) loss: 8.455233
(Iteration 9791 / 10000) loss: 5.499975
(Iteration 9801 / 10000) loss: 7.641086
(Iteration 9811 / 10000) loss: 8.572259
(Iteration 9821 / 10000) loss: 6.446636
(Iteration 9831 / 10000) loss: 6.343929
(Iteration 9841 / 10000) loss: 7.422457
(Iteration 9851 / 10000) loss: 6.309582
(Iteration 9861 / 10000) loss: 7.594930
(Iteration 9871 / 10000) loss: 7.485715
(Iteration 9881 / 10000) loss: 6.391306
(Iteration 9891 / 10000) loss: 6.614836
(Iteration 9901 / 10000) loss: 8.543337
(Iteration 9911 / 10000) loss: 6.584213
(Iteration 9921 / 10000) loss: 7.367630
(Iteration 9931 / 10000) loss: 7.050001
(Iteration 9941 / 10000) loss: 7.243069
(Iteration 9951 / 10000) loss: 8.637143
(Iteration 9961 / 10000) loss: 7.180451
(Iteration 9971 / 10000) loss: 6.506773
(Iteration 9981 / 10000) loss: 8.431390
(Iteration 9991 / 10000) loss: 8.480395
```




In [13]: `evaluate_model(capt_model)`

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-13-7af762d3b3a0> in <module>()
----> 1 evaluate_model(capt_model)


<ipython-input-11-d2c1609a6fc1> in evaluate_model(model)
     27         total_score = 0.0
     28         for gt_caption, sample_caption, url in zip(gt_captions, sample_captions, ur
---> 29             total_score += BLEU_score(gt_caption, sample_caption)
     30 
     31         BLEUscores[split] = total_score / len(sample_captions)


<ipython-input-11-d2c1609a6fc1> in BLEU_score(gt_caption, sample_caption)
      9     hypothesis = [x for x in sample_caption.split(' ')
     10                   if ('<END>' not in x and '<START>' not in x and '<UNK>' not in x)
---> 11     BLEUscore = nltk.translate.bleu_score.sentence_bleu([reference], hypothesis, we
     12     return BLEUscore
```

```
13
```

```
NameError: global name 'nltk' is not defined
```