Natural Language Processing
Assignment 4

Due April 8, 2019
12pts Total

0 Setup

Throughout this assignment, you will be implementing and analyzing different components of some image captioning architectures. To get setup to run the code, you will need to do the following:

1. Create an anaconda environment that has Python 3.6 or Python 3.7.
2. Using anaconda, install the following packages into this environment:
   - PyTorch 1.0
   - NumPy 1.16.2
   - h5py 2.9.0
   - Matplotlib 3.0.3
   - SciPy 1.2.1
   - scikit-image 0.14.2
3. Download the MS COCO captions and data splits from here: `http://cs.stanford.edu/people/karpathy/deepimagesent/caption_datasets.zip`.
4. We will not be using the entire MS COCO dataset. Instead, we will just download the validation set and work with subsets of the validation data as our train, validation, and test sets. Download the validation images from here: `http://images.cocodataset.org/zips/val2014.zip`.
5. Place the compressed caption JSON file and the compressed validation images in a directory of your choosing, then uncompress both directories. This will create 2 directories, `caption_dataset` and `val2014`. Make sure you put these 2 directories within another directory.
6. Run the preprocessing code with `python preprocess_data.py`. You can run the command with the `-h` option to view the command line arguments. For the assignment, you should leave any arguments with their default value, if they have them; otherwise, you need to provide them. You can try other preprocessing configurations if you're doing extra exploration, which is highly encouraged.

Now that you have everything downloaded and uncompressed, take a look at some of the images and their captions to get a better understanding of the dataset.

1 Vanilla RNN (2pts)

In this exercise, you will implement the forward pass of a vanilla RNN cell. We will assume that data is stored in batches with a batch size of N. You will first implement a single step of the RNN cell to get a feel for the basic operations. Then you will implement the full, multi-step RNN which is used in the rest of the assignment. The non-batch equation of the RNN is given by

$$h_t = \tanh(W_x x_{t-1} + W_h h_{t-1} + b),$$

where $x_t \in \mathbb{R}^D$, $h_t \in \mathbb{R}^H$. The learnable parameters of the LSTM are an input-to-hidden matrix $W_x \in \mathbb{R}^{H \times D}$, a hidden-to-hidden matrix $W_h \in \mathbb{R}^{H \times H}$ and a bias vector $b \in \mathbb{R}^H$.

1.1 Single Step (1pt)

Here you will implement a single step of a RNN. You need to implement the `__init__` and `forward` methods of `RNNDecoderSingle` in `src/layers.py`. The instructions in the code will guide you more on what to implement. Once you have an implementation, run the `test_rnn_single` function in `test_layers.py` to check your implementation. You should get an error of around 10^{-8} to 10^{-7}.

1.2 Multi-step (1pt)

Now you should implement the RNN for the entire input sequence. You need to implement the `__init__` and `forward` methods of `RNNDecoder` in `src/layers.py`. This should be a straightforward extension of the previous exercise. Once you have an implementation, run the `test_rnn_multi` function in `test_layers.py` to check your implementation. You should get an error of around 10^{-8} to 10^{-7}.

2 LSTM (2pts)

For this part, you will implement the forward pass of a LSTM cell for image captioning. Just as in the previous section, you will first implement a single step of the LSTM cell and then you will implement the full, multi-step LSTM. Again, the full, multi-step version will be used used in the rest of the assignment.

At each timestep we receive an input $x_t \in \mathbb{R}^D$ and the previous hidden state $h_{t-1} \in \mathbb{R}^H$. In contrast to the RNN, the LSTM also computes a 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 $a^{(t)} \in \mathbb{R}^{4H}$, which is given by

$$a^{(t)} = W_x x_t + W_h h_{t-1} + b.$$

We split this into four H-dimensional vectors, $a_i^{(t)}, a_f^{(t)}, a_o^{(t)}, a_g^{(t)} \in \mathbb{R}^H$, where $a_i^{(t)}$ consists of the first H elements of $a^{(t)}$, and so on. Next, we 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

$$i_t = \sigma(a_i^{(t)}), \qquad f_t = \sigma(a_f^{(t)}), \qquad o_t = \sigma(a_o^{(t)}), \qquad g_t = \tanh(a_g^{(t)}),$$

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

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \qquad\qquad h_t = o_t \odot \tanh(c_t)$$

where $\odot$ is the elementwise product. As before, we assume the data is stored in batches and we will work with transposed versions of the parameters. This allows us to efficiently compute activations $A^{(t)} \in \mathbb{R}^{N \times 4H}$ as $A^{(t)} = X_t W_x + H_{t-1} W_h$.

2.1 Single Step (1pt)

Here you will implement a single step of a LSTM. Just like with the RNN, you need to implement the `__init__` and `forward` methods of `LSTMDecoderSingle` in `src/layers.py`. Once you have an implementation, run the `test_lstm_single` function in `test_layers.py` to check your implementation. You should get an error of around 10^{-8} to 10^{-7}.

2.2 Multi-step (1pt)

Implement the LSTM for the entire input sequence. You should implement the `__init__` and `forward` methods of `LSTMDecoder` in `src/layers.py`. Run the `test_lstm_multi` function in `test_layers.py` to check your implementation. Again, you should get an error of around 10^{-8} to 10^{-7}.

3 Output layer

Implement a linear output layer in both your RNN and LSTM decoders. This layer should produce an $N \times V$ matrix, where V is the vocabulary size. Integrate this layer into your decoders, making sure to return the result of this layer, rather than the hidden states. Do not add a softmax function as this is handled for you in the rest of the code.

4 Attention Mechanism (2pts)

In this exercise, you will implement an attention mechanism and integrate it into the RNN and LSTM decoders. Specifically, you will implement the global, general attention (A.K.A. multiplicative attention) by Luong et al. (2015) (`https://aclweb.org/anthology/D15-1166`). In particular, you should refer to Equations 5, 7, and 8. For extra help on understanding the process of an (additive) attention mechanism, see the attention mechanism supplementary materials. Note, the supplementary materials covers the additive attention, not the multiplicative which is what you are implementing in this assignment. Once you're finished, you can test your implementation with `test_layers.py`. If your code is correct, doing so will give you an error of around 10^{-8} to 10^{-7}.

5 Image Captioning (4pts)

Try out both your implementations (RNN and LSTM, both with and without attention) on the MS COCO data. To do so, run `train.py` to train your model. Your implementation of the RNN decoder will be loaded by default. To train your LSTM implementation, make sure you add the argument `--decoder_cell_type lstm`. Train for at least 3 epochs and report your scores and plot the loss over the iterations. More training epochs should help improve your model, so you are

encouraged to do more epochs, if possible. Also, report the hyperparameters that you used (*e.g.*, decoder hidden size, learning rate, etc.). Note, make sure you uncomment the `reset_parameters` line in each of your decoder cells. Lastly, run `test.py` and report your best model's scores.

6 Analysis and Questions (2pts)

6.1 Attention Visualization (1pt)

Use the function in `src/visualize.py` to visualize the results of your attention mechanism on at least 6 images. Provide these images in your report and do a light analysis of your results.

6.2 Captioning Questions (1pt)

Let's look at the images with their captions in Figure 1. For each image in Figure 1, 3 captions are given. During training, each caption along with it's corresponding image is given to the model as a single training example. This means that the model will see the same image with multiple different captions during training. Examining these image-caption pairs, answer the following questions:

1. What is the reasoning behind having multiple reference captions per image? What is 1 advantage and 1 disadvantage from having multiple reference captions where each caption describes the image in a different way?

2. In images C and D, the names of the entities in the photos are missing from the captions. In general, what is one reason that it might be difficult to generate descriptions that incorporate the correct names? (Hint: Think about Zipf's Law.)

A **B** **C** **D**

	Caption
A	1) the old woman is feeding the birds in the street. 2) a person who is sitting on a bench with pigeons around them. 3) a courtyard, full of poles, pigeons, and garbage cans, also has benches on either side of it, one of which shows the back of a large person facing in the direction of the pigeons.
B	1) the sign states no swimming, water skiing or diving. 2) a sail boat traveling over a body of water. 3) the sign is posted on the beach for everyone to see.
C	1) a baseball player walking to his post with his bat. 2) a crowd watches on at a baseball game 3) a man walking across a field holding a baseball bat.
D	1) a number of people standing near one another wearing suits and ties. 2) a group of presidents standing next to each other. 3) our past five presidents are represented in this photograph.

Figure 1: Image captioning examples.