Programming Assignment 4:
Image Captioning

CSE 251B: Deep Learning
Winter 2021

Instructions

Due Saturday, February 27^{th}:

1. Start early! If you have any questions or uncertainties about the assignment instructions, please ask about them as soon as possible (preferably on Piazza, so everyone can benefit). We want to minimize any possible confusion about this assignment and have tried very hard to make it understandable and easy for you to follow.
2. Please hand in your assignment via Gradescope. We prefer a report written using LaTeXin NeurIPS format for each assignment. You are free to choose an alternate method (Word, etc.) if you want, but we still prefer NeurIPS format.
3. You should submit your code on Gradescope along with your report. For your own purposes, keep your code clean with explanatory comments, as you may want to reuse it in the future!
4. You will be using PyTorch (v 1.4) for this assignment as well. You should already know how to access the UCSD GPU server by this point but if forgotten, please see the material on Piazza to refresh your memory. Additionally, any updates or modifications to the assignment will be announced on Piazza.
5. Please work in teams of size 4-5. In extraordinary circumstances (e.g., you have COVID and are afraid of infecting your teammate), we will allow you to do it on your own (*however*, contrary to the QAnon rumor, COVID *cannot* be spread by Zoom!). Please discuss your circumstances with your TA, who will then present your case to me.
6. **Important**: For the group report, include an informative title, author list, and an abstract. The abstract should summarize briefly what you did, and the best percent correct you got on each problem. The report should be well organized with an introduction, background (if you review previous work), methods, results, and discussion for each programming part. Figures should be near where they are referenced, there should be informative captions on figures, clearly specified axes and figure keys, etc. Details of what to include in the the report along with rubric can be found at the end of the document.
7. **Note**: Training this model will take you about 1-2 hours, and may take longer if the GPUs on Datahub aren't free for usage. Any hyperparameter or architecture change that you make will take an hour before you figure out whether that change was correct or incorrect. So, start early!

Learning Objectives

1. Understand the basics of a recurrent neural network (LSTM).
2. Learn how to implement an encoder-decoder architecture in PyTorch for image captioning task.
3. Quantify the goodness of text generation by calculating BLEU scores.

Part I

Homework problems to be solved individually, and turned in individually

1. How do LSTMs deal with vanishing and exploding gradients? (2 points)
2. This question refers to Figure 1. Give an example of a domain or problem where the following types of RNN architectures could be used: (2 points each)
   - Figure 1b: many to one.
   - Figure 1c: one to many
   - Figure 1d: many in, then many out.
   - Figure 1e: Simultaneous many in, many out.

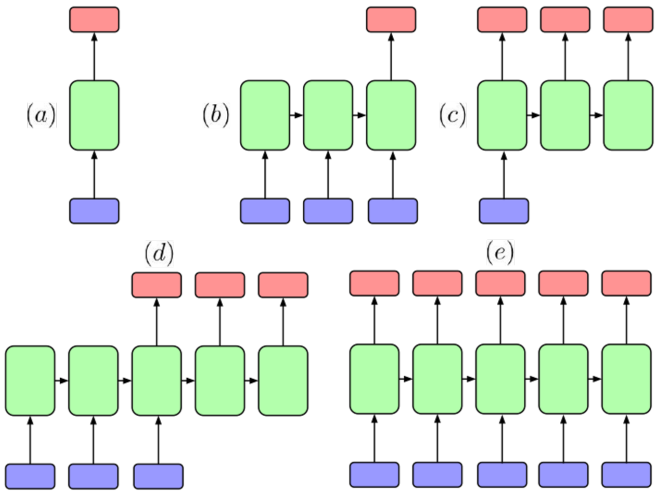

Figure 1: Basic Recurrent Network Architectures (except (a), which is feedforward!)

3. What is the function of each of the gates in an LSTM cell? (2 points)

Part II
Image Captioning using an LSTM network

1 Problem Description

In this assignment, we will explore the power of Recurrent Neural Networks to deal with data that has temporal structure. Specifically, we will generate captions for images. In order to achieve this, we will need an encoder-decoder architecture for this assignment. Simply put, the encoder will take the image as input and encode it into a vector of feature values. This will then be passed through a linear layer for providing the input to the LSTM. It will be trained to predict the next word at each step. The following figure illustrates this:

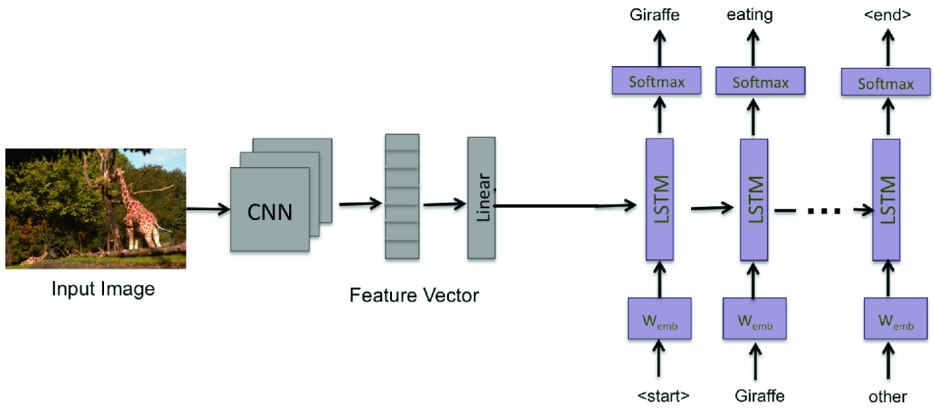

Figure 2: Encoder-Decoder architecture for image captioning (image credit: Deep Neural Network Based Image Captioning)

You will use a pre-trained convolutional network as the encoder and an LSTM model as the decoder. You will have to fine tune the encoder and train the decoder by backpropagating error into it. The error will come from caption generation. The training uses images and several captions for each image generated by humans. You will then run the network in generative mode to generate captions on images it has never seen before.

2 Dataset

2.1 Getting familiar with the data.

For the image captioning task, we will be using the dataset from the well-known COCO (Common Objects in Context) repository. COCO is a large-scale object detection, segmentation, and captioning dataset. The official website contains different versions for each of the tasks. To get an overall sense of the dataset, reading the official document (can be found here) is encouraged (but not required for our assignment). In our case, we will be using the COCO 2015 Image Captioning Task. This dataset already exists in the dsmlp cluster in the '/datasets/COCO-2015' directory. As the dataset is quite large, we will be using sub-parts of it for training and testing.

2.2 Preparing the dataset

For this assignment, you are given specific image ids for training and testing. As the original dataset is quite large, we will use about $1/5^{\text{th}}$ of it. Our training set contains around 82k images with roughly 410k captions while the test set has around 3k images with almost 15k captions. You have to first copy these images into your working directory. Apart from images, you will also need the captions for them, which you have to download from the server. The annotations are JSON files which contains image ids, image name, caption ids, captions etc. Fortunately, a notebook for creating train and test image datasets and downloading the annotations/captions is provided for you. You have also been provided with a split of training, validation and test datasets for tuning your hyper-parameters (size of embedding, number of hidden units, optimizer learning rate, etc.), so you don't need to do any splitting yourself.

One thing to note is that the code makes use of a 'COCO' object (imported from 'pycocotools.coco'). Please install the 'pycocotools' library if this has not already been installed (from the terminal: `pip install pycocotools --user`). Also, make yourself familiar with the different methods in this tool as they will play a critical role in completing the assignment. For your convenience, here are the descriptions of a few useful commands:

- `coco = COCO(caption_path)` : Creates a COCO object class which has useful methods. Here, `caption_path` is the path to the JSON file you downloaded using our script.
- `coco.imgToAnns[img_id]`: Returns a dictionary of all different captions of the given `img_id`.
- `coco.loadImgs(img_id)[0]['file_name']` : Returns the name of the image for the given `img_id`.

We have provided you the code to build the coco datasets and also configure different hyper-parameters. However, you should still try to understand how the COCO object works as you will need it to generate bleu scores.

3 Design the Network

3.1 Baseline Model:

This is the baseline model we introduced in Figure 2.

Note: The image embedding dimension after the Linear layer will have to be **exactly** the same as the hidden state dimension for LSTM.

3.1.1 Encoder:

This part of the assignment should look familiar to you because you will have to use a frozen pretrained convolutional network, namely ResNet50, as the encoder. You should, however, remove its last layer, and add a trainable linear layer which outputs a feature vector of a fixed size for each image. This sets the initial state of the LSTM network based on the image. You can augment the image input as in PA3, using any transformation of the original images you think suitable. Also, during development, you can downsize the images to make this encoding part faster, but note that this approach may lose useful details from the image. Once your network is working, you should then go back to the original size images. Note that the images in the dataset are of different sizes and aspect ratios, which we are resizing to 256x256 in the data loader. You can change this to any other dimensions you want.

3.1.2 Decoder:

For decoding, we will train a recurrent neural network to learn the structure of the caption text through **"Teacher Forcing"**. Teacher forcing works by using the teaching signal from the training dataset at the current time step, $target(t)$, as input in the next time step $x(t+1) = target(t)$, rather than the output $y(t)$ generated by the network. A couple of suggestions/guidance on this part:

- In the first timestep, the decoder takes the encoded image to initialize its hidden state and the '<start>' word as input. Then we feed the caption words one by one and use teacher forcing to train the model. Each word is first one-hot encoded based on the vocabulary and then further converted to a lower dimensional embedding using a fully connected layer. This layer is trained by backprop (a word encoder). We will also need a fully connected layer from the hidden states to the output, to generate the one-hot encoded output.
- You will need to create the vocabulary of all the different words in all of the training captions, i.e., create a data structure to efficiently get the index of a word and vice-versa. The code for vocabulary creation is provided to you. This structure will be used in the datasets to build the one-hot encodings and also to generate string captions from your model's outputs.
- You should use the available Pytorch modules to build the LSTM network. In short, Pytorch's LSTM takes three inputs: the *current feature*, hidden state (h_t) and cell state (c_t). We can combine both h_t and c_t as tuple and call it "hidden states". As shown in Figure 2, the *current feature* for the first step should be the embedding of '<start>' word. Hidden states for this step should be initialized from encoded image embedding, ie, both h_0 and c_0 are to be set as image embedding from encoder. In this step the output will be the first word in the current caption (e.g., 'Giraffe' as shown in the figure), and so on. We only use the hidden state to generate the output. Refer to this link for more details.

- To use '<start>' or any subsequent word as *current feature*, get its index from the vocabulary you created, convert it to one-hot vector and pass it through the linear embedding layer to convert it to the expected input dimension for the LSTM. You can also take advantage of Pytorch's nn.Embedding which does one-hot encoding + linear layer for you.
- For implementation convenience, you will want to 'pad' the captions to convert them into fixed length so that they can be vertically stacked in a mini-batch. The code for this is provided to you.
- For choice of loss function, you can either use Pytorch's *NLLLoss* (negative log likelihood loss) for which you should use a softmax layer over the outputs/predictions or you can use *CrossEntropyLoss* which automatically combines both softmax + NLLLoss. To get the loss for validation/testing, pass your validation/test data in the same manner as your training data (i.e. teacher forced). The only difference here is that you would have learning turned off. You will need these losses for your report.
- You should evaluate your model on the training and validation set after every epoch or after certain number of mini-batches to prevent overfitting.
- It is also a good idea to print out the generated captions after every epoch (complete passes through the dataset) or a certain number of steps (number of mini-batches completed) to see if the generated captions are improving and making sense. Two approaches to generating captions are described below.

3.1.3 Generating Captions:

In the generation stage, the idea is to "let the network run on its own", predicting the next word, and then use the network's prediction to obtain the next input word. There are at least two ways to obtain the next word

- **Deterministic:** Take the maximum output at each step. This usually does not work well at all.
- **Stochastic:** Sample from the the probability distribution. To get the distribution, you need to calculate the weighted softmax of the outputs: $y^j = exp(o^j/\tau)/\sum_n exp(o^n/\tau)$, where o^j is the output from last layer, n is the size of the vocabulary, and τ is the "temperature". The temperature controls how stochastic the sampling is: As the temperature approaches 0, the distribution is nearly deterministic, as it goes to infinity, the distribution is completely uniform. With this approach, you should get a different caption each time (unless the temperature is too low).

For the generated captions, you will also need to evaluate BLEU-1 and BLEU-4 scores by comparing them to actual captions. The code to evaluate BLEU scores is provided to you.

3.2 Model variations:

3.2.1 Vanilla RNN vs LSTM

Replace the LSTM module in your encoder with a vanilla RNN. Compare the training/validation curves, and test BLEU scores for the 2 implementations (LSTM and vanilla RNN). Use the same set of hyper-parameters (hidden units, optimizer, learning rate etc.) for both models. Discuss your findings.

3.2.2 Tuning the embedding size and the hidden size of the LSTM

Your baseline LSTM decoder uses words as input by converting them to a fixed-size feature vector/embedding. Try to tune the values of the embedding size and the hidden size to find which values give you a better performance.

Remember that the output size of the Linear layer after the image has been encoded will be of the same dimension as the hidden state dimension in the baseline model. You are expected to vary this embedding size/hidden state dimension and report the results.

Compare the performance (in terms of train/val loss and test BLEU-1 and BLEU-4 scores) of this model with your baseline models.

4 Report:

1. Title: The title has to be informative and not generic like '251B PA4 Report'. Also you have to include author's list. (1 point)
2. Abstract: A short description of what you did. Please mention any key findings, interesting insights, and final results. (5 points)
3. Introduction: The introduction should introduce and motivate the problem, not repeat the abstract. Why is this problem interesting? (Hint: people can do it...) How will you approach it - without details on your approach! Things like the number of hidden units, etc. do not go here! (5 point)
4. Related Work: Cite any paper/slides for the methods you have used for this project. Also you can cite any Pytorch documentation that helped you implement the PA. (2 points)
5. Methods section: In a subsection titled "Dataset", describe the dataset. Then, in a subsection, entitled "Model", describe models you will use: the architecture for both baseline(LSTM) and vanilla RNN model. Describe how you sample outputs from the decoder and how your model obtains word embeddings. In your description, please report all the hyperparameters for your best models only. Then, in a "Results" section, provide a plot of your training loss and validation loss vs number of epochs. How is the learning curve of the vanilla RNN model compared to the baseline (better or worse)? Does vanilla RNN need more epochs to achieve satisfactory performance or less? Provide your reasons. (30 points)
6. For each of your best models, also report the cross entropy loss on the test set. Discuss how the vanilla RNN model performed compared to baseline. *(5 points)*
7. Generate captions for both of your best models in the deterministic approach (i.e. max output) for the test set and report BLEU-1 and BLEU-4 scores against the reference captions. You may use any library function that implements BLEU scores. Helper functions for these are provided in the code, but you are free to use any other library function if you wish. Which model was able to achieve better score? Discuss your results. *(5 points)*
8. For the baseline model, experiment with the stochastic approach and try different temperatures (e.g.0.1, 0.2, 0.7, 1, 1.5, 2, etc.) during generation. Report BLEU-1 and BLEU-4 scores for at least 5 different values. Is there a range that works better than deterministic approach? Why or why not? Provide your reasoning. *(10 points)*
9. Compare the baseline model with one that you found after fine-tuning the embedding size and the hidden size. Describe your choice of hyperparameters. Report train/validation loss curves and test BLEU-1 and BLEU-4 scores for this model. Compare and discuss performance with the baseline model. *(10 points)*
10. Using your best model across all the experiments that you did (baseline model, varying the embedding size, architecture 2), visualize the generated captions along with some images. Take any random set of images from the test data, predict your caption for that, and plot the predicted caption along with the original captions (Note that there will be multiple original captions for each image). From these results, plot 5 images along with the predicted caption and the original captions where you think your predicted caption makes sense (ones for which the model is performing well). Also Plot 5 images where the model is performing poorly. (5 points)
11. Team Contribution: A short paragraph from each team member with what they contributed to the project - team members won't necessarily get the same grade if someone slacked off! (1 points)

5 References

BLEU scores:

1. https://en.wikipedia.org/wiki/BLEU
2. https://www.aclweb.org/anthology/P02-1040.pdf
3. https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

6 Sample results

Examples of correct results

```
Actual captions:
desk sits in corner next to the couch with a computer monitor and keyboard on top
a room with a desk and computer in it
a home office area with many pictures on the wall
a desktop computer sitting on top of a wooden desk
a chair and a computer in a room


Predicted caption: a room with a chair and a television on it
```

Figure 3: Example - 1

```
Actual captions:
set of ripening bananas on a plant with a teddy bear placed atop
a teddy bear sitting atop bunches of green bananas
a large bunch of green bananas with a bear in them
a teddy bear sitting in a bunch of bananas on a banana tree
a stuffed bear has been placed in a bunch of bananas


Predicted caption: a bunch of bananas hanging from a tree branch
----------------
```

Figure 4: Example - 2

```
Actual captions:
a very tall building has a clock on the front of it
a bird flying near a clock tower in a city
a large church steeple with a clock on it
a very tall clock tower next to other tall buildings
a large tall brick tower has a clock on top


Predicted caption: a large building with a clock on the top of it
-----------------
```

Figure 5: Example - 3

Examples of incorrect results

```
Actual captions:
a woman holding shears in one hand a tile in the other
a person standing holding a large pair of scissors
the woman in the blue shirt is using really big scissors
a smiling woman holding a large pair of scissors
a woman in a blue shirt holding a pair of large scissors


Predicted caption: a man in a white shirt is holding a pizza
-----------------
```

Figure 6: Example - 1

```
Actual captions:
birds perched on logs on a lake near a wall
a log in the water with three birds perched on it
a group of water birds sitting on drift wood in a body of water
several birds sitting tree branches sticking out of the water
birds on branches sticking out of a body of water


Predicted caption: a man on a surfboard in the water
-----------------
```

Figure 7: Example - 2

```
Actual captions:
a closed umbrella sitting on a balcony ledge above traffic
an umbrella caught on an upper floor railing
a close up of an umbrella on a hand rail
a umbrella laying on the edge of a railing
the umbrella is stuck in the fence area


Predicted caption: a person holding a umbrella in front of a building
-----------------
```

Figure 8: Example - 3