Topics in Artificial Intelligence (CPSC 532S): Assignment #4: *Image Captioning or Retrieval*

Due on Friday, February 15, 2019 at 11:59pm PST

In this assignment you will combine components you built in Assignment 2 and 3 to produce either an image captioning **or** image/sentence retrieval system. Note that you have a choice of working on one or the other, **you do not need to do both**. You will find that the captioning model is simpler to implement (at least steps 1–5) given the components you have worked on already.

Programming environment

You will be using **PyTorch** for this assignment. If you are using Microsoft Azure credits provided, you will need to provision a Data Science Virtual Machine (VM) to use for the assignment. To do so, please follow the instructions here (clickable link):

Provisioning of the Data Science Virtual Machine for Linux on Microsoft Azure

If you provisioned the VM for the previous assignment, **DO NOT DO THIS AGAIN**; re-use the VM you created. PyTorch should be pre-installed and available in the Azure Data Science VM. **Make sure you are keeping track of your Azure credits.** Remember that you are being charged as long as VM is running, even if it is idle and not executing any jobs. It maybe best to do some of the coding and debugging locally (in CPU mode) and run on the VM only when you verified that your code works and you need GPU support. If you opt to do this assignment on your personal computer, it is your responsibility to install PyTorch and ensure it is running properly. A number of tutorials for doing this are available on-line.

Data

To train the model you will be using both images (same 20K training images we used for Assignment 2) and corresponding descriptive sentences (20K sentences in total we used for Assignment 3) from MS-COCO dataset. Note that for each training image there are 5 descriptive sentences annotated by different people on Amazon Mechanical Turk in COCO; however, we are using only 1 sentence per image to make the learning easier. For testing you will use 100 validation images and corresponding 500 sentences (again from Assignment 2 and 3).

Assignment Instructions

You will need to follow instructions in the Jupyter notebook and submit the assignment as saved Jupiter notebook when done (including the results of executing code). As part of the assignment we are providing pre-processing code for both images (same as Assignment 2) and sentences (same as Assignment 3). In this assignment you have a choice of implementing one of the following two models (**for full credit you are only required to implement Part 1 or Part 2, not both**).

Part 1: Image Captioning

Here you will build an image captioning model by combining a pre-trained VGG-16 image encoder from Assignment 2 with LSTM-based language decoder from Assignment 3. Effectively you are implementing the following paper:

- Show and tell: A neural image caption generator, O. Vinyals, A. Toshev, S. Bengio, D. Erhan, CVPR, 2015.

which requires the steps 1–5 detailed below. You will then be augmenting the model with self-attention. In the process, you will learn about attention models and marginally improve the captioning performance.

1. **Setup Image Encoder.** You will need to load pre-trained VGG-16 model with the weights trained on ImageNet. You will need to get rid of `softmax` as you did in Part 3 of Assignment 2, so you will end up with `fc2` layer producing $\mathbf{x}_i \in \mathbb{R}^{4,096}$ feature encoding for a given image i.

2. **Setup Language Decoder.** Here you will start with the language decoder model that you used as part of the end-to-end network in Part 6 of Assignment 3. You will need to pass image encoding $\mathbf{x}_i$ as the hidden state input into the first LSTM cell (i.e., $\mathbf{h}_0 = \mathbf{x}_i$). However, this would only work if the hidden state is dimension of the $4,096$, which is way too high dimensional. In order to get a more reasonably dimensional representation you will need to insert a linear layer to project from $\mathbb{R}^{4,096} \rightarrow \mathbb{R}^{300}$ (note that Vinyals paper used 512-dimensional hidden state and image feature projection, so you could also use 512 instead of 300, which will likely be a bit better). You will also need to do the same for the cell state. The remainder of the language decoder, including the CrossEntropy loss should remain identical to the Assignment 3.

3. **Training Captioning Encoder-Decoder Architecture.** For training the model you can follow the training procedure in Assignment 3, by training on $(\text{image}_i, \text{sentence}_i)$ pairs. However, you will want to freeze the Image encoder (not back-propagate through it). You can relax this and optimize/fine-tune end-to-end once the language decoder is at, or close to, convergence. You are not required to fine-tine the model end-to-end, though this will likely improve the results a little.

4. **MAP and Sampling Inference.** We will reuse the decoder inference functions from Assignment 3. The only minor change that is needed is providing the first hidden state from VGG-16 as opposed to getting it from the language encoder.

5. **Measuring Performance.** For validation images compute the average BLEU score. Since for each image there are 5 ground truth sentences, you will need to supply the same MAP inference for each of 5 ground truth sentences when computing the BLEU score.

6. **Self-attention Model.** Construct a one-`fc`-layer neural network f_{att} such that $e_{i,j} = f_{att}(\mathbf{a}_{i,j})$. The input to this network should be 512-dimensional vector $\mathbf{a}_{i,j}$ and the output should be 1-dimensional vector $e_{i,j}$. You will form the input by taking the output of the last convolutional layer of VGG-16 network (after MaxPool2d) which has dimensionality of $512 \times 7 \times 7$ and feeding each of 7×7 cell's representation as input. Note, you should re-use the VGG-16 network that is in memory instead of instantiating a new one (otherwise you will likely run out of memory). Pushing all cells through this `fc` layer will result in a $7 \times 7 = 49$-dimensional vector $\{e_{i,j}\}$. Now apply `softmax` to this resulting vector, obtaining:

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{7}\sum_{m=1}^{7}\exp\left(e_{k,m}\right)}.$$

In the above α's are known as *attention weights*; because f_{att} is only considering the image itself to predict cell's importance, this is called self-attention. Now, given the attention weights lets construct the context vector:

$$\mathbf{c} = \sum_{i=1}^{7}\sum_{j=1}^{7} \alpha_{i,j}\mathbf{a}_{i,j}.$$

The last part is to augment the captioning model built above to accept as input, at every LSTM decoder cell, not only the embedding of the previous word token but also the computed context vector $\mathbf{c}$. This will effectively change the LSTM cell's input from 300 dimensional (assuming 300 dimensional embedding of words) to $300 + 512 = 812$-dimensional. Everything else should remain the same. You will need to re-train the decoder along with the attention network; again keeping VGG-16 fixed at least for a period of a time during training.

7. **Measuring Performance of Self-attention Model.** Recompute the BLUE scores as in Step 6 but with this new self-attention model. Also, for a few images in the validation set visualize the attention when computed in the forward pass during evaluation. You can display attention by simply taking the 7×7 attention map and upsampling it to the size of the original image; then visualizing `image` $\times$ `attention` (computed as a per color channel element-wise product). For slightly better visualization normalize the attention map before upsampling by dividing all values by `max` attention.

Part 2: Image/Language Retrieval

Here you will build a image-language retrieval model. The model is simplified variant of the following paper:

- Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models, R. Kiros, R. Salakhutdinov, R.. Zemel, CoRR abs/1411.2539 (2014)

although many other papers use very similar formulations for variety of tasks. Note that this option is more difficult than Part 1 above, so I would only suggest doing it if you feel comfortable with the material and previous Assignments.

1. **Setup Image Encoder.** You will need to load pre-trained VGG-16 model with the weights trained on ImageNet. You will need to get rid of `softmax` as you did in Part 3 of Assignment 2, so you will end up with `fc2` layer producing $\mathbf{x}_i \in \mathbb{R}^{4,096}$ feature encoding for a given image i.

2. **Image Embedding.** To obtain an image embedding, on top of `fc2` layer of VGG-16 we need to add a linear layer (you can also use an embedding layer in PyTorch) that projects image features to a joint embedding space: $\mathbb{R}^{4,096} \rightarrow \mathbb{R}^{300}$. In other words, the feature vector in the embedding space is $\mathbf{v}_i = \mathbf{W}_I \cdot \mathbf{x}_i$, where $\mathbf{W}_I \in \mathbb{R}^{300\times 2,096}$ are parameters of the layer to be optimized.

3. **Setup Language Encoder.** Here you will start with the language encoder model that you used as part of the end-to-end network in Part 6 of Assignment 3. You will need to make sure your end-to-end encoder-decoder from Assignment 3 works well and load pre-trained parameters for the language encoder. We will take the last hidden state of the encoder $\mathbf{h}_{last} \in \mathbb{R}^{300}$ as the encoding of the sentence.

4. **Language Embedding.** We will assume that there is identity mapping between the sentence and the joint embedding space, so the representation of the i-the sentence in the embedding space is simply $\mathbf{s}_i = \mathbf{h}_{i,last}$ (note that we could add an embedding layer, i.e., linear layer that would project $\mathbf{h}_{last}$ to an arbitrary dimensional joint embedding space).

5. **Defining Distance Metric in the Joint Embedding Space.** Note, the way we defined image and language embedding, they both live in the same 300 dimensional space. We will need to define a distance measure on this embedding space to train the model and perform retrieval. The easiest, and among very effective, is dot product of normalized vectors (similar to cosine distance), defined:

$$d(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}}{||\mathbf{x}||} \cdot \frac{\mathbf{x}'}{||\mathbf{x}'||}. \tag{1}$$

6. **Defining the Loss to Optimize.** In retrieval, our loss will be based on pairwise ranking. The idea is simple. Given ground truth pair $(\text{image}_i, \text{sentence}_i)$, the distance in the embedding space between elements of that pair should be smaller than the distance of image_i to any other sentence and, similarly, smaller than distance of sentence_i to any other image (these pairings are called contrastive, i.e., non-descriptive, pairings). Therefore the loss can be defined using the following equation that can be computed based on two ground truth pairs $(\text{image}_i, \text{sentence}_i)$ and $(\text{image}_j, \text{sentence}_j)$ (where $i \neq j$):

$$\mathcal{L}(\theta) = max\{0, \alpha - d(\mathbf{v}_i, \mathbf{s}_i) + d(\mathbf{v}_i, \mathbf{s}_j)\} + max\{0, \alpha - d(\mathbf{v}_i, \mathbf{s}_i) + d(\mathbf{v}_j, \mathbf{s}_i)\} \tag{2}$$

   where θ includes parameters of the two encoders and embeddings. A typical value of α is 0.2 and this loss can be implemented using **HingeEmbeddingLoss** in PyTorch by supplying $d(\mathbf{v}_i, \mathbf{s}_i) - d(\mathbf{v}_i, \mathbf{s}_j)$ and $d(\mathbf{v}_i, \mathbf{s}_i) - d(\mathbf{v}_j, \mathbf{s}_i)$ as parameters (in two separate calls and summing the two losses).

7. **Training the Model.** With model and loss defined, training takes a standard form. Notably, one requires two image-sentence pairs to compute the loss, not just one. Also, I strongly advise to freeze both encoders and only optimize parameters of the Image Embedding at least as the first step. End-to-end fine-tuning is possible after Image Embedding converges, but is not required.

8. **Testing Performance.** Given a trained model, we can rank any set of images with respect to a language query and vice versa (rank any set of sentences with respect to the image query) by simply computing the distance between sentence and each of the images in the set in the embedding space using $d(\mathbf{x}, \mathbf{x}')$ or vice versa.

   Let's consider doing this for sentence retrieval (captioning) in validation set first. Given a query image, we can embed it into the joint embedding space using VGG-16 feature and projection through the matrix $\mathbf{W}_I$. We can also embed all 500 sentences in the validation set into the joint embedding space by running them through the encoder and picking up last hidden state. We can then compute 500 distances corresponding to the pairings. Note that 5 of the sentences will correspond to the ground truth. We can measure how good our model is by computing average rank of those ground truth sentences. A random model should give an average rank close to 250; an optimal model would give an average rank of 3 by assigning ground truth sentences to the top-5 ranked spots. With a well trained model we should expect to get an average ranking somewhere in-between. Once we get average rank for the ground truth sentences, we can also average those (average) ranks across all 100 image queries in our validation set. Please report this average as part of the assignment. Similar procedure can be done to measure performance of image retrieval (image search) given a sentence query.