CSE 253: Programming Assignment 4
Image Captioning

**Winter 2020**

# Instructions

## Due Sunday, March $1^{st}$:

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 everybody 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. 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 as to refresh your memory. Additionally, any updates or modifications to the assignment will be announced on Piazza.
3. Please work in teams of 4 or 5 individuals (no more than 5).
4. Please submit in your assignment via Gradescope. The report should be in NeurIPS format or another "top" conference format (e.g., IEEE CVPR, ICML, ICLR) - we expect you to write at a high academic-level (to the best of your ability). Make sure your code is readable and well-documented - you may want to reuse it in the future. Your report should be written as if you are writing a real conference paper.

# 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.

# Image Captioning using 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. The decoder will take this output from encoder as hidden state and starts to predict next words at each step. The following figure illustrates this:

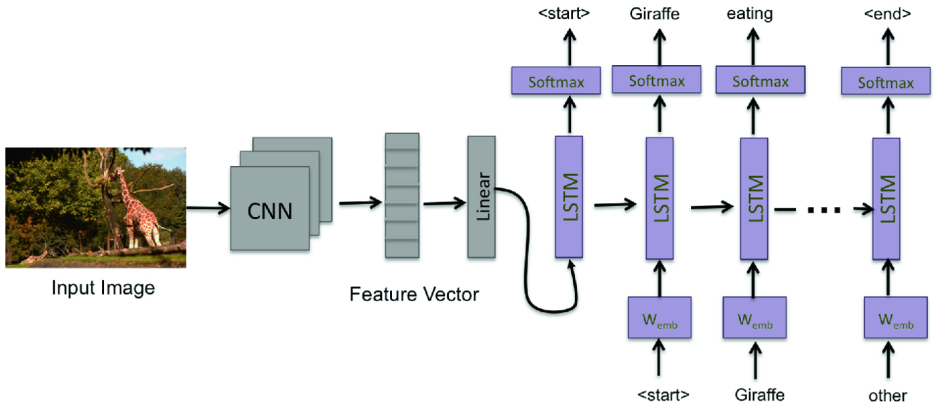


Figure 1: Encoder-Decoder architecture for image captioning (image credit: Deep Neural Network Based Image Captioning)

You will have to use a pre-trained convolutional network as the encoder and an LSTM model as a choice of decoder. You will have to fine tune the encoder for the given image and train the decoder using words extracted from their captions and then run the network in generative mode to generate captions.

## 2 Dataset

### 2.1 Getting familiar with the data.

For image captioning task, we will be using dataset from the well-known COCO (Common Objects in Context) repository. In general, 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. As only training and test splits have been provided to you, feel free to extract from the training set a validation set of your choice for tuning your hyper-parameters (size of embedding, number of hidden units, optimizer learning rate, etc.).

One thing to note is that the notebook 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`.

# 3 Design the Network

## 3.1 Baseline Model:

This is the baseline model we introduced in Figure 1.

### 3.1.1 Encoder:

This part of the assignment should look familiar to you because you will have to use a pretrained convolutional network, namely ResNet50, as the encoder. You should, however, remove its last layer, and add a trainable linear layer to fine tune it for our task. The encoder should output a feature vector of a fixed size for each image. Like PA3, you are encouraged to use any transformation of the original images as you think suitable. Also, you can resize the images to lower dimensions to make this encoding part faster, but note that this approach may lose useful details from the image. You may want to do this during development, and then go back to the original size once you have your model designed and working. Note that the images in the dataset are of different sizes and aspect ratios, so you'll have to randomly crop a square section to use as input to the encoder.

### 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:

- 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. Make use of different methods from the 'COCO' object as you see fit.
- You should use the available Pytorch modules to build the LSTM network. In short, Pytorch's LSTMCell take three inputs: the *current feature*, hidden state ($h_0$) and cell state ($c_0$). We can combine both $h_0$ and $c_0$ as tuple and call it "hidden states". As shown in Figure 1, the *current feature* for the first step should be the output of encoder to predict '<start>' word. Hidden states for this step should be set to None. Then in the second step '<start>' will be the input and the output will be the first word in the current caption (e.g.'Giraffe' as shown in the figure), and so on.
- 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 a linear layer to embed into a feature (or you can 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 in a mini-batch to convert them into fixed length.
- For choice of loss function, you can either use Pytorch's *NLLLoss* for which you should use a softmax layer over the outputs/predictions or you can use *CrossEntropyLoss* which 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.
- **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 $\tau$ is the "temperature". What's interesting about this is you should get a different caption each time.

Save the caption for each of the test image ids to a file. You will need this file to calculate various scores for your report.

### 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 Using pre-trained word embeddings

Your baseline LSTM decoder uses a words as input by converting them to a fixed-size feature vector/embedding. Currently your network learns these word embeddings during training. In this experiment, you will use pre-trained word embeddings such as Word2Vec or GloVe. If you use Pytorch's nn.Embedding layer, you can initialize its weights with a matrix containing pre-trained word embeddings for all words in your vocabulary, and freeze the weights (i.e. don't train this layer).

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

## 4 Report:

1. Describe 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. Also 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. (14 points)

2. For each of your best models, also report the cross entropy loss and perplexity score on the test set. Discuss how the vanilla RNN model performed compared to baseline. *(6 points)*

3. 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. For this, an optional helper notebook is provided for you (but you are not required to use it). Which model was able to achieve better score? Discuss your results. *(6 points)*

4. 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. *(7 points)*

5. Compare the baseline model with one that uses pre-trained word embeddings. Describe which word embeddings you chose to use and why? Report train/validation loss curves and test BLEU-1 and BLEU-4 scores for the model using word embeddings. Compare and discuss performance with the baseline model. *(7 points)*