Machine Learning Engineer Nanodegree

Capstone Proposal - Image Captioning

Raiymbek Akshulakov

June 2nd, 2020

Proposal

Domain Background

A human would probably look at this photo, and instantly come up with tens different thoughts on what is depicted in there. A few examples would include: two friends walk under the trees, two guys walk in the park, a person with a guitar walks

near another person. A quick glance for a second is enough to make such captions for humans. However, what is easy for our brain to do is sometimes very hard to do for machines. Automated text description generation is the task called image captioning. The application of such technology is endless. For example, an image descriptor for the visually impaired because this way the application can not only detect the objects in front of a person but also understand the relations among the objects.

The task of image captioning is the process of ongoing research. The novel approaches to image captioning include various RNN models. The use of those RNN models was inspired by the success in the task such as machine translation which is basically a quite similar task to image captioning. Examples of those models would be a paper *"Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models"* (https://arxiv.org/pdf/1411.2539.pdf) that used encoder-decoder RNN model and also *"Show, Attend and Tell: Neural Image Caption Generation with Visual Attention"* (https://arxiv.org/pdf/1502.03044.pdf) that added an attention mechanism to encoder-decoder RNN model for image captioning. For Udacity Capstone project I will build a simple LSTM seq-to-seq model to be able to generate image captions to a given image

Problem Statement

The goal is to build a model that takes an image as input and generates the text description of it.

Step:

- Find the appropriate dataset containing images and captions to them
- Build a model the model for image captioning
- Train the model in google Colab using free GPU
- Evaluate the model

The final model is expected to be able to take image from internet and generate the captions to it

Datasets and Inputs

For the project I will use for the Coco dataset

Takes 10 hours and 20 GB. Data links:

- train images http://msvocds.blob.core.windows.net/coco2014/train2014.zip

- validation images http://msvocds.blob.core.windows.net/coco2014/val2014.zip

- captions for both train and validation
http://msvocds.blob.core.windows.net/annotations-1-0-3/captions_train-val2014.zip

Each image in the dataset contains 5 captions. It contains a training set for training and a validation set for evaluations.

Training and Validation annotations are stored in two big json files. The file can be seen as a huge python dictionary - let' say **data.** Then **data['images']** is a list of all image information such as the name, id, height, width, and the link for image and **data['annotations']** is a list of all annotations. Each annotation element has the id of the image and a caption.

Solution Statement and Benchmark Model

I will be using the LSTM seq-to-seq model. The model can be divided in two stages - encoder and decoder. Encoder takes the data that we want to generate a caption from,(in our case an image) and gets a feature vector from it. For image data it is the best to use a CNN for feature extraction. In fact, instead of training my own CNN I will use an already pre trained and established ***InceptionV3*** model. I will take the last layer before the last Dense layer in the model. Next, decoder is an RNN model that takes the encoder's feature vector as the initial hidden state. The decoder model tries to predict the next word given the current word. We start with a START token and recursively repeat the process until we get an END token or until we reach some length. Here the captions serve both as input and ground truth of the model - each word tries to predict the next one. During generation model, the model will recursively plug the predicted word

Evaluation Metrics

I will be using three different metrics here in addition to just simple validation loss:

- Using the trained model it will be possible to get the so-called "probability" of the caption. It can be calculated by multiplying all the softmax probabilities from the decoder of each word in the caption. Averaging it over 5 captions and all the validation set will give the probability of the whole validation set. Also I can use the accuracy of the next predicted word for each word in the validation set.

- Another evaluation metric is widely used for comparing generated sentences to a reference sentence - Bilingual Evaluation Understudy Score, or BLEU for short. It can be implemented with nltk python library.

- Another evaluation metric from nltk python library is meteor score.

Project Design

- The very first step is to prepare the training images for training. It will be hard to upload the dataset to google colab every time because it is a very large dataset(20 GB). My plan is to divide the dataset into several smaller parts, convert each part to h5 format and upload to google drive from the local machine. In the Google Drive, using colab get each part and run it through the encoder and store it in the google drive(also in parts). Once it is done the initial data can be deleted, and now the encoder feature vector's weight will be a lot smaller than previous 20GB

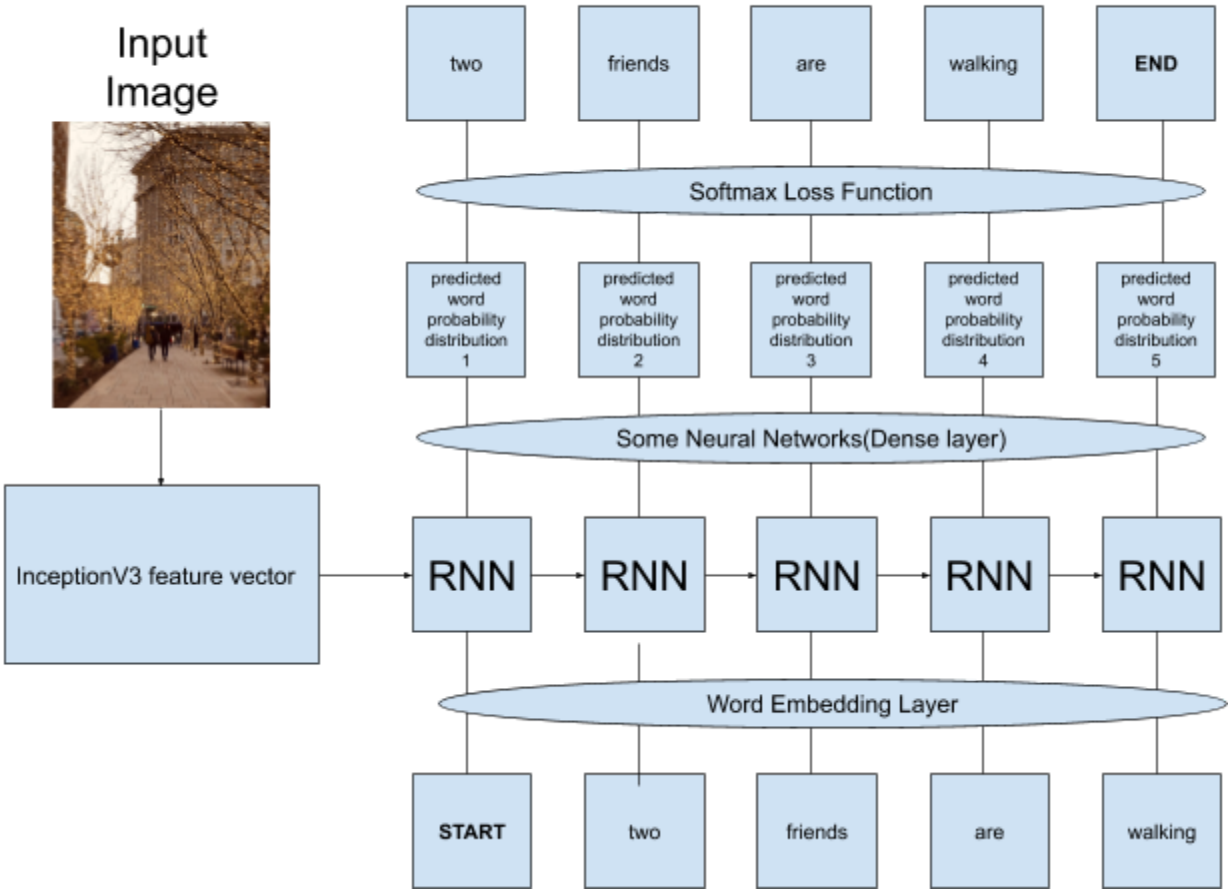
- Once we get the encoder results, before training the decoder I will need to preprocess the caption data. In this case there is no need for stemming, lemmatization, and removing stop words because we need almost all the variations of all words. First step would be to tokenize all the captions. To do that , I will get rid of very rare words and change them with UNK tokens(less than 5 or 10 occurrences). I will put a START token at the beginning of each captions and END token and the end. For batching purposes I will also need to use padding for all the captions to one length with PAD token.

- The exact architecture of the decoder is subject to change because I want to try different architectures on the train set. However the essence of the model will remain the same as explained in the Solution section. It will be an ImageCaptioning class containing all the architecture:

    - It will be able to train the model using the *train_model* method.

- Save and restore the model once trained
- Generate a caption for given image
- Evaluate the model on the test set

Training Process



Generation Process