COL774 Assignment4

Aniket Gupta
2019CS10327

Hardeep Kaur
2019CS10354

November 2021

1 Introduction

This assignment involves developing a model for reading captions embedded in an image. We used Encoder-Decoder Architecture for modelling the problem. An encoder converts the input image into its feature vector representation and a decoder is applied on this vector representation to generate the sequence auto-regressively (one word/character at a time). There are two parts in this assignment - a non competitive part and a competitive part. The encoder model uses a Convolutional Neural Network (CNN) and the decoder model is based on RNN/LSTM.

The CNN based encoder extracts the features from the images by passing it through different convolution and max-pool layers. For the decoder, we have used LSTM (Long Short Term Memory) based model instead of a simple RNN model. A simple RNN model takes the output at its previous time step as its current input, stores it in its memory for a short period of time and give the next output based on this. The problem with a simple RNN is that it fails to store information for a long period of time. It is here LSTM comes to rescue. The architecture of LSTM stores in its memory for a longer time and can capture relations over larger distances.

The overall process can be divided in the following steps:

- Pre-processing Step
  - Image Pre-processing
  - Caption Pre-processing
- Encoder Model
- Decoder Model
- Inference
- Evaluation

The pre-processing step is same for both the competitive as well as non-competitive part. The only difference between the two parts is in the encoder model. For the competitive part, we have used pre-trained ResNet50 model for the encoder whereas for the non-competitive part, we trained VGG16 model from scratch for the encoder. The decoders in both the parts have the same architecture.

2 Pre-processing Step

2.1 Image Pre-processing

The input images were of different sizes. We resized all the images to 224 x 224 pixels. We then normalised the pixel values to Mean = (0.485, 0.456, 0.406) and Standard Deviation = (0.229, 0.224, 0.225). The three values in the tuple are for the three channels. These values are in accordance with the mean and standard deviation values used by the pre-trained models. Normalising the images improved the BLEU score obtained by a large margin.

2.2 Caption Pre-processing

Initially we tried to remove punctuation and perform stemming, but that decreased the BLEU score. Thus, we didn't perform stemming and punctuation removal in our final model. Since the decoder model would need to process texts of different lengths, we added <start> and <end> token for each sentence. Also, during the pre-processing, we prepared a vocabulary dictionary that contained all the tokens in the training dataset. Each token is then assigned a unique integer value. The decoder model is given a tensor of integers and at each step, it returns an integer value. The integer value is matched in the dictionary to get the correct token. During the training, captions in each batch are padded to the maximum length of the caption.

3 Encoder Models

3.1 Competitive Part

For the competitve part, we tried using pre-trained VGG16, VGG19 and ResNet50 models. Among these three, ResNet50 gave the best BLEU score. We replaced the last layer of the ResNet50 model with a fully connected layer. The output of the ResNet50 model is passed through a ReLu Layer. The ResNet50 model takes an input image of size 224*224*3 and returns a feature vector of length 256. The best results obtained for all the models we experimented with in this part are:

Model	**Epochs**	**BLEU score**
VGG 16	9	0.1023
VGG 19	10	0.0922
ResNet50	12	0.1455

3.2 Non-Competitive Part

For the non-competitive part, we trained VGG16 architecture from scratch. We also tried with VGG19 architecture, but it was performing worse on the given dataset. The architecture is same as a standard VGG16 architecture except the last layer. We replaced the last layer with a fully connected layer. The best results obtained for all the models we experimented with in this part are:

Model	**Epochs**	**BLEU score**
VGG 16	13	0.12074
VGG 19	12	0.08067

4 Decoder Models

The architecture of the Decoder Model is same for both the competitive as well as non-competitive part. During the training, all the caption words are passed through an embedding layer which gives gives a fixed length output vector which is fed to the LSTM model. The PyTorch package packs the sequences in a batch to send it together to the LSTM layer. The PyTorch LSTM module takes care of the teacher-forcing and sends shifted inputs to the cells accordingly. Teacher forcing helps in quick learning of the LSTM module. In teacher forcing, the RNN uses ground truth for the previous time step as input, instead of model output from a prior time step. The output of the LSTM is passed through a linear layer for the final output. At each step, the linear layer returns a probability distribution over the tokens in the vocabulary.

5 Inference

For making prediction on a given input test image, we first pass the image through the encoder. The encoder returns a fixed length vector containing features of the images related to the texts embedded in it. The output vector of the encoder is then passed through the decoder. At each time step, the decoder returns a distribution over the tokens in the vocabulary and the new state of the model. This new state acts as input for the next time step.
There are following two methods to select the token at each time step.

- **Greedy Search** : In Greedy Search, we chose the token which has the maximum probability in the distribution returned by the LSTM layer at each time step. This can be represented as:

$$\arg\max_{y} \prod_{t=1}^{T} P(y^{<t>}|x, y^{<1>}, ..., y^{<t-1>})$$

- **Beam Search Algorithm** : In Beam Search Algorithm, we recursively take top few words (equal to beam size) and generated a tree of most probable captions. Beam Search uses breadth-first search to build its search tree. If k is the beam width, it progresses level by level and moves downwards only from the best k nodes at each level. In other words, it generates all the successors of the current level's state at each level of the tree. However, at each level, it chooses only best k number of states for next level. Other nodes are not taken into account. We emperically found the optimal beam width to be 3 while trying different values in range 1-10.

6 Hyper-parameters

The BLEU score changed significantly on varying the different hyperparameters. The core of all learning lied in the hyperparameter tuning of the model for better convergence and performance. The following discussion shows the different values of different hyperparameters that we tried.

- **Learning Rate:** For the competitive part, we used learning rate of 0.05. We tried with smaller learning rates but it made the convergence very slow. For the non-competitive part, we used a learning rate of 0.1. We took larger learning rate in non-competitive part because we had to train the encoder from scratch. With smaller learning rate, the VGG16 CNN model was taking a lot of time to converge. Thus, we increased the learning rate to 0.1 for this part.

- **Dropout:** For the competitive part, we have used a dropout layer just before the last layer of the encoder with dropout probability of 0.5. For the non-competitive part, we have used a dropout layer between the last two layers of the encoder and one dropout layer between the output layer of the encoder and input layer of the decoder with dropout probability of 0.5.

- **Embedding and Hidden State Dimensions:** Embedding Dimension is the constant size to which all images are encoded into by the Encoder. Hidden state dimension is the size of output produced by the hidden layers of LSTM. We have kept the Embedding and Hidden State Dimensions to 256.

- **Batch Size:** We tried with different batch sizes in $\{8, 16, 24, 32\}$. For the competitive part, 32 gave the best BLEU score whereas for non-competitive part, batch size of 8 helped the model to converge faster.

7 Evaluation: BLEU score

We have used character level BLEU score for evaluating the performance of the model. Thus, n-character in the following discussion refers to a group of n consecutive characters. The BLEU score is the precision

score of the generated caption against a reference caption. While comparing the captions, BLEU algorithm looks for n-character matches in the two sequences and evaluates the strength of match with precision score. The precision score is the fraction of n-characters in the caption that also appear in the reference. For each n-character size, a character in the reference caption cannot be matched more than once. Also, since small sized perfect matches would achieve a 1.0 precision, we impose a brevity penalty so that such captions are not considered good.
Note: While calculating the precision score for length n, all the character groups up to length n are considered. The precision score is calculated as:

$$p_n = \frac{\text{number of matched n-chars}}{\text{number of n-chars in output caption}}$$

The brevity penalty is added as below:

$$\beta = e^{min(0,1-\frac{\text{len}_{\text{ref}}}{\text{len}_{\text{output}}})}$$

where len_{ref} is the length of reference caption and len_{output} is the length of output caption generated by our model.
The geometric weighting for the precision of n^{th} character is given as follows:

$$w_n = \frac{1}{2^n}$$

The BLEU score is then given by:

$$BLEU = \beta \prod_{i=1}^{k} p_n^{w_n}$$