Indian Institute of Technology Madras

CS6910 Deep Learning

Assignment 3

Vimal Suresh Mollyn ED17B055
Dyava Naveen Reddy ME17B140
Ramakrishnan NA18B030

April 24, 2022

Vimal Suresh Mollyn, ED17B055
Dyava Naveen Reddy, ME17B140
Ramakrishnan, NA18B030

Contents

1 Task 1 & 2

1.1 Introduction to the task

The name of the task is automatic image captioning. In this task we have to generate captions of given images using an Encoder-Decoder framework. The model should take input as image, extract important features from the image, and then output a text sentence describing the image.

1.2 Pipeline for Image Captioning

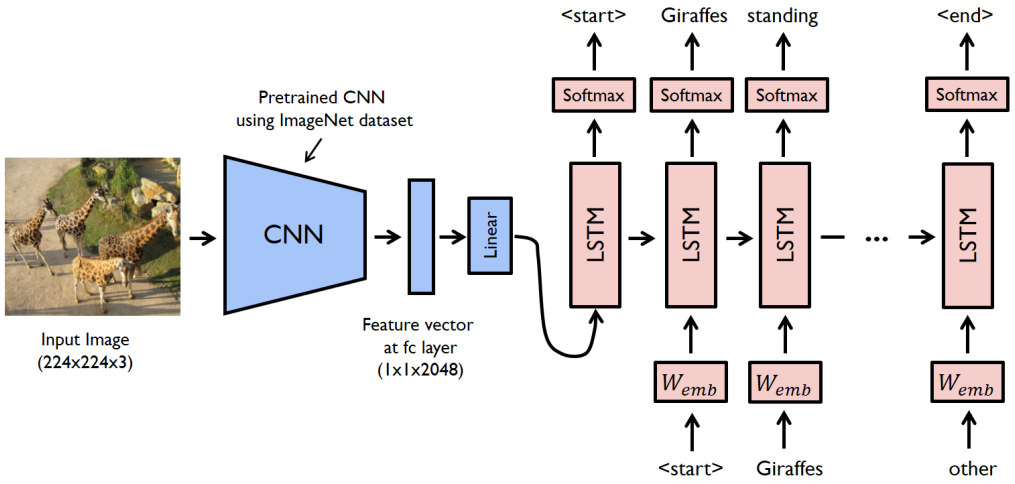

Figure 1: The Image Captioning pipeline

The pipeline for this task consists of two important steps. The first step is to pass the input image through a CNN feature extractor. The features extracted from the CNN feature extractor are further passed through a NetVLAD for feature aggregation. When the image is processed through CNN & NetVLAD the encoder part is done. Next the features extracted from the encoder are passed through a decoder framework, typically RNN based models are used to generate the text output. In this assignment a vanilla RNN & a LSTM model is used for the decoder architecure.

1.3 Data Pre-Processing of Images to use them in CNN-Encoder

The Image data given doesn't have a common size among images. So they have to be pre-processed such that all the images will have same resolution. In this task we

have pre-processed images such that they have 400 x 400 as dimension. The images given are colour images, so Thieu have 3 Chanels. The final size of the image to be input to the CNN model is 400 X 400 X 3.

1.4 Data Pre-Processing for Text data (Captions)

The caption data is a text data. These captions are of various lengths. We have to clean the text first before we apply other procedures. First of all all the punctuation marks in the captions are removed. Keeping the punctuation marks will make the model training more complicated. After the punctuation marks are removed all the text is converted to lowercase. Then in the next step the captions are tokenized using spacy tokenizer. Tokenization is the task of splitting a text into small segments, called tokens. The tokenization can be at the document level to produce tokens of sentences or sentence tokenization that produces tokens of words or word tokenization that produces tokens of characters.

The tokenizer is usually the initial step of the text preprocessing pipeline and works as input for subsequent NLP operations like stemming, lemmatization, text mining, text classification, etc.

In Spacy, the process of tokenizing a text into segments of words and punctuation is done in various steps. It processes the text from left to right. First, the tokenizer split the text on whitespace similar to the split() function. Then the tokenizer checks whether the substring matches the tokenizer exception rules. For example, “don’t” does not contain whitespace, but should be split into two tokens, “do” and “n’t”, while “U.K.” should always remain one token. Next, it checks for a prefix, suffix, or infix in a substring, these include commas, periods, hyphens, or quotes. If it matches, the substring is split into two tokens.

After tokenizing the tokens have to be converted into corresponding word embeddings so that it can be input to the LSTM/RNN model. To Convert the tokens into GloVe is used. To be specific the GloVe embedder which is trained on 840B tokens, 2.2M vocab is used. The size of embedding dimension for each toked is 300. So each word will be converted into a 300 dimensional vector. Glove in itself has lot of words in its vocabulary. However we have added additional works into the vocabulary like <sos> to represent start-of-sentence, <eos> to represent end-of-sentence, <unk> to represent unknown tokens and <pad> to represent padding tokens. Every token in the vocabulary has corresponding token indices. So every token must be converted to corresponding token indices and then the token indices must be passed through

the the embedding layer to convert the tokens into corresponding embeddings.
An important thing that has to be done before pre-processing is to pad the sequences. In order to train the Neural Networks faster, we have to batch them. But the captions data has varying lengths of sentences. In order to mitigate this issue padding must be done. A general procedure is to pad all the sentences to the maximum size of the sentence in the given data. But a more efficient way to do is to pad each sentence to the maximum size of the given sentence in a given batch only. This makes the training efficient by reducing the unnecessary computation through the <pad> tokens.

1.5 CNN Encoder Model

For the given task it was enforced to use the CNN model from the assignment. The CNN model from the previous assignment has kernels of size 3x3, stride of 1 in the convolutional layers, mean pooling with a kernel size of 2x2 and stride of 2 in the pooling layers, 4 feature maps in CL1. The number of feature maps in CL2 is 5. The model had 2 layers.
The outputs of the feature vectors obtained from the CNN are passed through the NetVLAD module and at last 60-feature vectors of dimension-5 are flattened and passed to the encoder. After flattening we will have 300 features.

1.6 LSTM & RNN Decoder Module

The LSTM & RNN both are trained with BPTT method. Each token in a sentence is passed one after another sequentially. while training initially the feature map extracted from the encoder is passed and then start token is passed, followed by the remaining tokens. The output of the hidden layer of LSTM/RNN is passed through a linear layer followed by a softmax layer to predict the token index. Then the cross-entropy loss is used to calculate the loss.

1.7 Results

Some of the example captions are provided below.

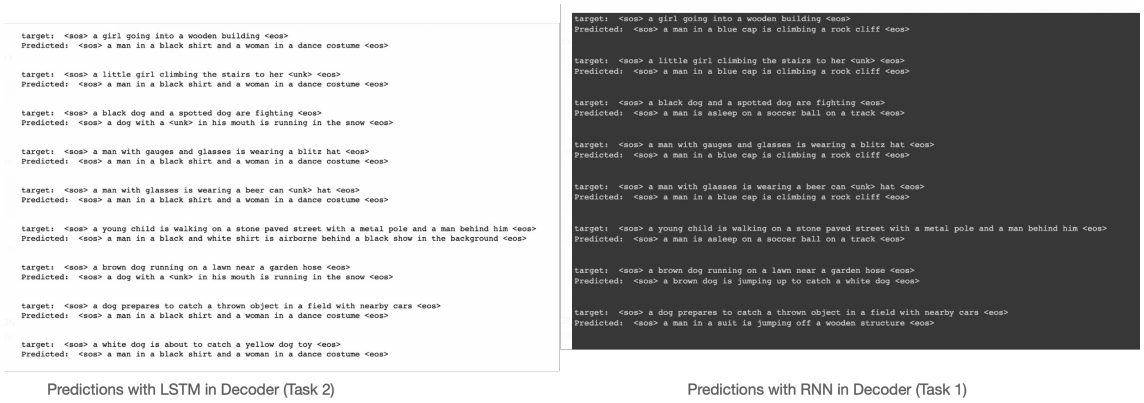

Figure 2: Example captions from the Image Captioning Task

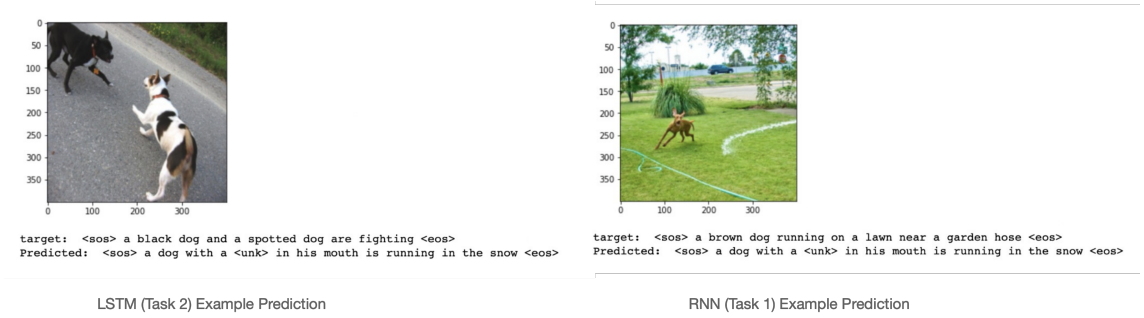

Figure 3: Example captions from the Image Captioning Task LSTM and RNN

2 Task 3

2.1 Introduction

In this problem, we were tasked with the problem of neural machine translation (from English to Tamil) using an LSTM based encoder decoder model. In order to do this, we proposed to use the general *seq2seq* model pipeline, which we will now describe.

2.2 Neural Machine Translation

The general task of neural machine translation is the following. Given a source sentence in language 1, we would like to know the same sentence in a different language (language 2). To achieve this, state of the art methods employ *Encoder-Decoder* models. One such popular model is *seq2seq* which involves the use of a recurrent neural network in order to train the model.

Given a sentence, it is first *tokenized* into a set of words/subwords. Then, in order to feed it into a Neural network, the words are transformed into a vector embedding space, through an embedding layer. In this assignment, we chose to use a popular *pretrained* word embedding know as *GloVe* (or Global Vectors for Word Representation).

Now we have a sentence split into tokens, each represented by a vector of a certain dimension (we chose the 300-D vector representation). Next, these tokens are passed into an Encoder network, whose aim is to capture the semantics and meaning of the sentence. In practice, we implement this encoder network as an LSTM, using the pytorch framework. The result of this encoding process is a so called *context vector*, which is of the same dimension as the hidden state of the LSTM. This is then fed into another *Decoder* Network, which performs translation.

The decoder network is also constructed as a LSTM in practice. In particular, we tokenize the target sentences (tamil) into subword token embeddings, which are further fed into the Decoder network. Tokens and embeddings are extracted using the *indicBERT* model, through the *huggingface transformers* library. The outputs of the LSTM are then fed into a linear layer that maps hidden states to vocabulary, which we then use a softmax on top of in order to get the predicted word.
Finally, both these models are combined together into a single *seq2seq* model, which we train end to end, using Backpropagation through time. This is also illustrated in figure 4.

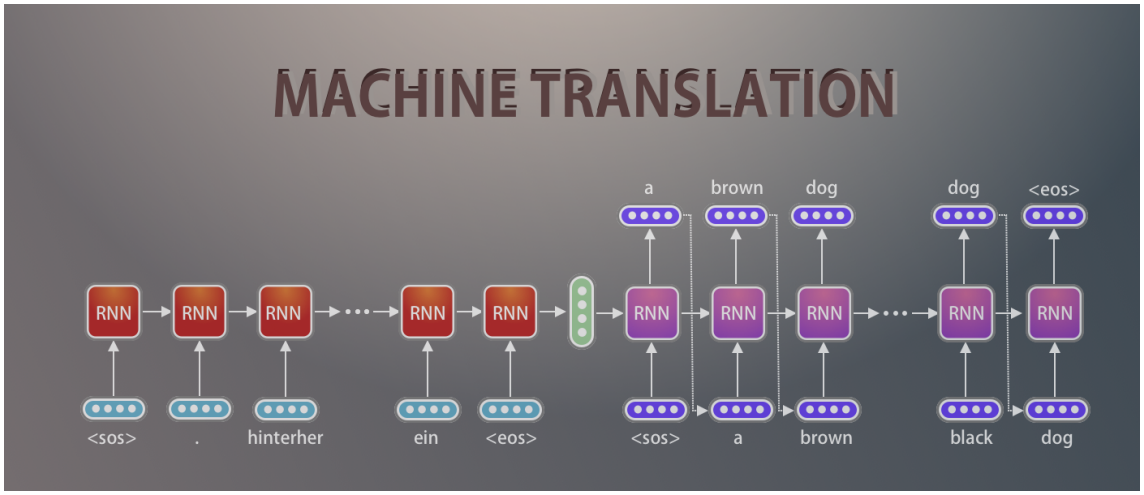

Figure 4: The Seq2Seq model

2.3 Data Preprocessing

We perform some data preprocessing steps, listed below:

1. Strip all the punctuation from the sentence
2. Convert all the letters to lowercase
3. Tokenize the english sentences with the *spacey* tokenizer
4. Initialize the GloVe Vectors and include a special <pad> and <unk> token, along with their initial values.
5. Create a dataloader, with a padding function in order to obtain same length sequences

2.4 Training procedure

We now describe the procedure for training a neural machine translation system. First, we pass the source sentence recursively into our encoder model, so as to obtain a hidden *context* vector. Next, we feed this hidden context vector along with a *<start>* token to the decoder, and ask it to make a prediction. This prediction is further fed back into the recurrent network, repeatedly, until the max sequence length is achieved. Finally, loss is calculated as a sum of the individual losses wrt to each of the target words in the target sentence, using the cross Entropy loss. Thus, in effect, this problem is a multiclass classification problem. We used the Adam optimizer with a learning rate of 3e-4 in order to train the network. The Loss vs timestep is plotted in figure 5.

During inference/ test time - a source sentence is fed into the encoder, thus extracting a hidden context vector. This vector is further fed into the Decoder along with the same *<start>* token and a predicted word is extracted. This predicted word is then fed into the LSTM repeatedly, until the *<stop>* token is seen or the max sequence length is seen.

2.5 Hyperparameters

While one could theoretically experiment with various hyperparameters, our hardware limited us to the use of a relatively small network, which didn't allow for much experimentation. Details of our model hyperparameters can be found in the code.

Figure 5: The Training loss for the Seq2Seq model

2.6 Results

Find below a table of sample english sentences and their corresponding tamil translations. The BLEU scores are also provided.

3 Note of Training Resources

Unforunately, we did not have access to extensive training resources such as GPUs. As a result, our models took an very long time to train and often did not lead to good results (as in the machine translation task). To the best of our knowledge, it seems like if we trained the models for longer, we would have got much better results. We request you to please take this into consideration while grading.