Generation of Captions for Memes

C M Rakin Haider
Department of Computer Science
Purdue University
West Lafayette, IN 47906
chaide@purdue.edu

Abstract

Images that convey particular meaning in online interactions are commonly known as memes. Memes are popular among young generation as a method to convey short jokes. Memes mostly contain an image that sets up the context and a text that conveys the actual joke. In this project, we implement a caption generation method for meme image templates. We use an encoder-decoder based image caption generation method proposed in [6]. In this method there is an encoder that generates and image embedding and a decoder which takes the image embedding and generates a new text. We trained several models with different hyper-parameters. Due to long training time, the models were only trained for 10 epochs. After this short training the trained models were not able to generate meaningful texts for the images.

1 Introduction

The last decade has witnessed and unprecedented rise in the popularity of deep learning due to availability of large datasets and increased computation capacity. Deep learning techniques have demonstrated outstanding performance is several areas which was previously considered as hard problems. The development and introduction of large scale convolutional neural networks (CNNs) has achieved high performance in image classification. On the other hand, recurrent neural networks (RNNs) such as LSTMs, GRUs are known to perform well in handling sequential data such text, audio etc. New advancements of generative adversarial networks (GANs) have also made it possible to produce realistic computer generated samples. Automatic image captioning combines the capabilities of both CNN and RNN to generate new captions for images. In this project, we focus mainly on generating captions for a subset of images which are known as memes.

Memes are a common form of communication in social networks and online forums such as Facebook, Reddit etc. Memes usually contain an image that has a well-known meaning accompanied by a short text to convey a message. More often memes are used by young generation in informal setup. However, since memes are concise form of communication their use are often found in formal context as well. The popularity of memes have created the need of automatic meme generation (often known as memefication) from image templates. Examples of memes can be seen in Fig. 1.

In this project, we implemented a known method of image captioning commonly known as Show and Tell proposed in [6]. Show and tell follows an encoder-decoder approach. The basic idea of show and tell is to generate a representation of the image using a CNN. Then this representation is fed in to an RNN to generate related textual representation which is known as caption. In this project, we use a well-known CNN architecture, Inception-V3 [4] to generate image representation. Then we use LSTM [1] model to generate textual captions. The ouput of the CNN is fed to LSTM as the first element of the sequence to generate context variables of LSTM. Then, the element corresponding to *start* is fed into the LSTM model to obtain the first word of the sequence. Then the first word is fed back in to the model and this process is repeated until the end of sentence is generated.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

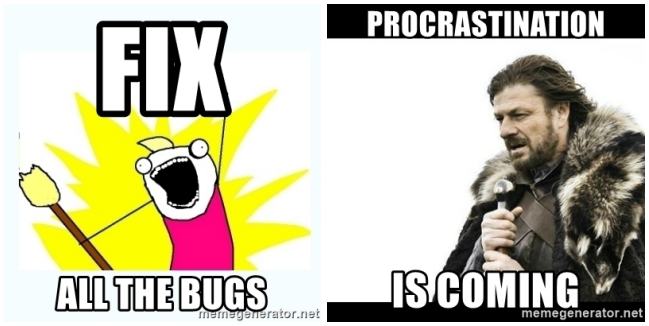


Figure 1: Example of Memes

The meme dataset contains a set of images and for each image it contains multiple captions. We varied the hyper-parameters to train multiple models. However, the training process requires a huge amount of time. Therefore, we had to consider a subset of the dataset for training and only 10 epochs for each model. The resulting models weren't able to generate meaningful captions. In the following section, we discuss the methodology, experimental results and our comments about the results.

2 Methodology

In this section, we discuss the methods that are implemented as part of this project.

2.1 Show and Tell

Show and tell is a well-known image captioning method proposed in [6]. Fig. 2 shows the pictorial presentation of the method. Show and tell contains two parts, the first one in encoder and the second portion is a decoder. We discuss about each in the following paragraphs.

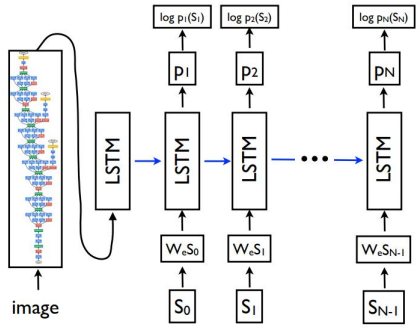


Figure 2: Pictorial representation of Show and Tell method collected from [6]

2.1.1 Encoder

Convolutional neural network(CNN) is a category of neural network that is designed to exploit spatial information of each pixel in image recognition. There are four major types of layers in convolutional neural networks. Most common types of layers in CNN are convolutional layer, non-linear activation layer, pooling layer and finally fully connected layer and softmax layer for classification. The convolutional layer applies filters over patches of the image to generate feature maps. The purpose of non-linear layer is to introduce non-linearity since it is seen that most real-world data have at least some degree of non-linearity. Pooling layers are used to reduce dimensions while at the same time retain most significant information. The output from convolutional layer or pooling layer is flattened and fed into a linear layer. The final linear layer usually contains one node for each class and it is usually followed by a softmax layer to generate classification probabilities.

In our implementation, we used the state-of-the-art pretrained Inception-V3 [4] architecture. It was trained on ImageNet dataset. The model contain 3 convolutional layer, 1 pooling layer, 3 convolutional layer, 3 inception layers, one pooling, one linear and one softmax layer. This model introduced inception layer which trains several filters of several sizes so that the network can decide which filter size is most appropriate for this task. The model also has an auxiliary network which pushes important gradients to the lower layer so that the model can combat vanishing gradient. Since we will not train the inception-v3 architecture we do not discus about auxiliary model here.

Our encoder model loads the pretrained Inception-v3 model except the last linear layer (and the softmax layer). Rather the input to the last linear layer is fed to a new linear layer, which is optimized in the training process. The output of the new linear layer is the representation of the image. Let, X is an image, and $INC(x)_{-1}$ is the input to the last layer of Inception-v3, then the representation Γ of X can be defined as,

$$\Gamma(X) = MLP(INC(X)_{-1})$$

2.1.2 Decoder

The decoder should be a RNN. RNN is a nerual networ architecture which can take sequence of inputs. In RNN, each element of the input sequence is fed into the RNN and for each element the RNN produces a hidden unit and an output. Each hidden unit depends of the hidden unit of the previous element and the current input element. As the decoder, we have used Long Short-Term Memory (LSTM) architecture. The forward pass of LSTM

$$\begin{aligned}
z_t &= tanh(W_Z x_t + R_z h_{t-1} + b_z) \\
i_t &= \sigma(W_i x_t + R_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f x_t + R_f h_{t-1} + b_f) \\
c_t &= z_t \odot i_t + c_{t-1} \odot f_t \\
o_t &= \sigma(W_o x_t + R_o h_{t-1} + b_o) \\
h_t &= tanh(c_t) \odot o_t
\end{aligned}$$

Here, W_z, W_i, W_f, W_o are input weights, R_z, R_i, R_f, R_o are recurrent weights, b_z, b_i, b_f, b_o are bias terms and x_t is the t^{th} element of the input sequence.

In this method, we first feed an image to a CNN to generate a representation for the image. Then this representation is fed into the LSTM to generate the hidden variable h_0 and context variable c_0. Then each element of in the sequence $x_1, x_2, \ldots x_n$ are fed into the LSTM one by one to generate the hidden variables. These hidden units are then fed to a linear unit which converts the hidden unit into a vector whose size is the length of the vocabulary. This output is passed through a softmax layer to obtain the probability of each word in the vocabulary to appear at timestep t. The process can be summarized as follows,

$$\begin{aligned}
h_0, c_0 &= LSTM(\Gamma_{img}(X), \bar{0}, \bar{0}) \\
h_t, c_t &= LSTM(\Gamma_{word}, h_{t-1}, c_{t-1}) \\
y_t &= argmax(softmax(MLP(h_t)))
\end{aligned} \tag{1}$$

Here, Γ_{img} is the image representation generated from CNN and Γ_{word} is a word embedding. In the implementation of the decoder we do not apply the softmax after the last layer. This is due to the fact that PyTorch `CrossEntropyLoss` automatically includes `LogSoftmax` and `NLLLoss` inside it.

If the training dataset is $\mathcal{D} = \{(X^i, Y^i)\}_{i=1}^N$, where X^i is the i^th image and Y^i is the i^th caption (sequence of word in the vocabulary). The loss of the optimization process is defined as follows,

$$\begin{aligned}
\mathcal{L}(D) &= \sum_i^n \sum_j^{|Y^i|} -log(P(y = Y_j^i | \Gamma(X), Y_{1:j-1}^i)) \\
&= \sum_i^n \sum_j^{|Y^i|} -log(softmax(y_t)_{Y_j^i}) \qquad \text{(From 1)}
\end{aligned}$$

2.2 Glove embedding

Word embedding is a representation of each word such that words with similar meaning will have similar embeddings. Word embedding can be trained by considering words and nodes of a graph where each edge (u, v) corresponds to whether the word u appears in the k-word context of the word v. Then if we train the node embedding of this graph we will obtain a word embedding for each word. The skipgram model to train word embedding can also be defined as learning the first order markov chain using the following equation,

$$p(X_{t+1} = x|X_t; U, V) = \frac{exp(< U_x, V_{X_t} >)}{\sum_{x'} exp(< U_{x'}, V_{X_t} >)}$$

Here, U and V are parameters of the neural network. In this project, we use pre-trained Glove embedding which is trained on words from Common Craw with 42 billion tokens [3].

3 Experimental Results

In this section, we discuss about the experiments performed for this project. The codes and the dataset for this project can be found in `https://github.com/rakinhaider/MemeCaptionGenerator`.

3.1 Dataset

The dataset contains 2509 meme templates. Each meme template has about 160 captions. The dataset was collected from the repository [2] of the authors of [5]. The dataset in total contains 414388 image-caption pairs. We start with the set of captions and remove contractions from the dataset. To do this we use `pycontractions` library. It replaces the contracted words like 'you've', 'didn't' with 'you have', 'did not' etc. Pycontractions works in three stages. First, it replaces any contraction that has only one rule. Then, for contractions with multiple possible replacements it generates all possible replaced texts, performs grammar check and finds the closest sentence to the original sentence exploiting word embeddings. After removing contractions, we removed all the punctuations using the list `stirng.punctuation`. Finally, we build the vocabulary by tokenizing each caption. We put each word in the caption in the vocabulary. The vocabulary simply contains two dictionary which maps each word to an index and vice versa. We also remove any word from the vocabulary which has lower than a certain number of appearance in the captions. The vocabulary also includes 4 special words, i.e. `<unk>, <start>, <end>, <pad>`. Any word not in the vocabulary is considered as `<unk>`.

3.2 Training

We randomly shuffled the dataset and used only 50000 samples from the dataset for training. Using the entire dataset requires a long time in the GPUs. We experimented with the following hyper-parameters. The default values for batch-size and learning rate are 1028 and 0.0001 respectively.

Hyper-parameter	Values	Default
Embedding Size (image and word)	50, 100, 200, 300	300
Hidden Unit Size	50, 300, 500	50
LSTM layers	2, 3, 4	3
Vocabulary Threshold (Minimum appearance in dataset)	2, 3, 4	2

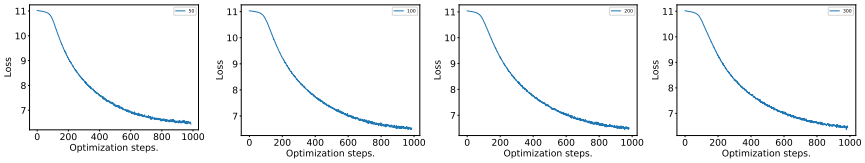
Table 1: Hyper-paramters for training

Each model was trained for 4 hours in GPU (maximum 10 epochs).

Since we do not allow fine tuning of inception-v3 parameters, to speed up training process, we pre-generated all the outputs from inception-v3 by feeding all the images. Later in the encoder we simply loaded the pre-generated outputs as embedding that maps image_id to embedding. To make this possible the image labels are also mapped to indices. Finally, the outputs of the embedding layer is fed into a `torch.nn.Linear` layer which maps the output of the inception-v3 network to a vector of size of $\Gamma(X)$.

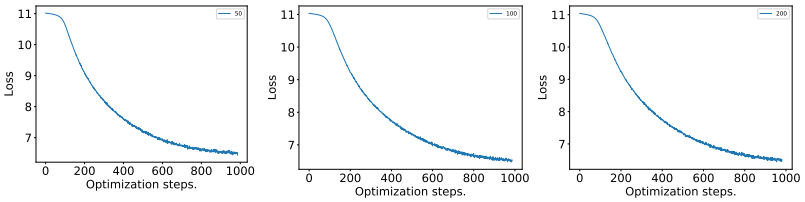
In decoder, we could allow train embedding for the vocabulary. Instead, we chose to use pre-trained Glove embedding. Any word that is already in the Glove vocabulary used its Glove embedding, but words which are not in the vocabulary uses random embedding vector. This embedding is loaded in a `torch.nn.Embedding` model. The index of each word is fed to the embedding model to obtain its embedding, alter this embedding is fed into the LSTM model. We use `torch.nn.LSTM` in our implementation.

In the following figure we, show the learning curves of loss after each optimization step in the training for different values of the hyper-parameters.



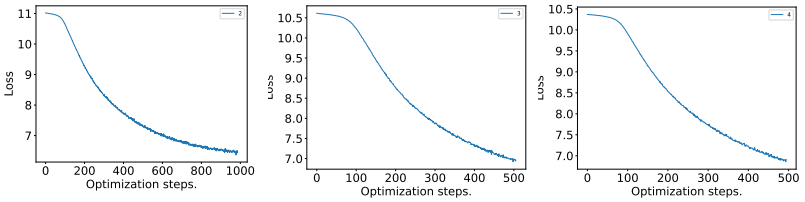
(a) Embedding size 50 (b) Embedding size 100 (c) Embedding size 200 (d) Embedding size 300

Figure 3: Loss vs optimization steps for each embedding size.



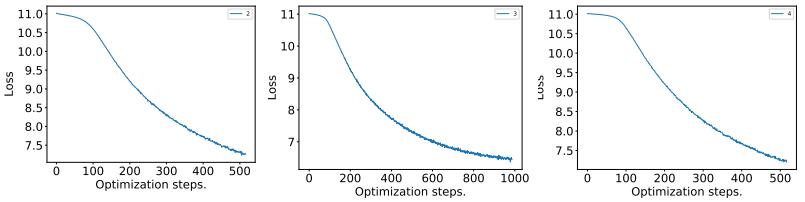
(a) Embedding size 50 (b) Embedding size 100 (c) Embedding size 200

Figure 4: Loss vs optimization steps for each size of hidden unit of LSTM.



(a) Threshold 2 (b) Threshold 3 (c) Threshold 4

Figure 5: Loss vs optimization steps for each minimum threshold of occurrence of word in vocabulary.



(a) Layers 2 (b) Layers 3 (c) Layers 4

Figure 6: Loss vs optimization steps for each number of LSTM layers.

The figures show that the loss values were decreasing. A longer training period might be better to obtain better models. Similarly, we could also choose to allow training the embeddings rather than loading pre-trained ones which could have trained a better representation of the images and words,

but could leed to higher training time. Every time we achieved a lower loss that previous lowest loss value, we checkpoint both encoder and decoder. After the training method we pick the last checkpoint as the best model.

3.3 Sampling

The sampling method is performed by the decoder. If the decoder is provided the representation $\Gamma(X)$, it uses the representation to generate h_0, c_0 by feeding it to the LSTM. Then the decoder obtains the embedding of `<start>` and feeds it through the LSTM along with h_0, c_0. The generated hidden variable is forwarded through a linear layer and a softmax to generate predictions for the next word. We pick the word with the highest probability as the predicted word from the vocabulary. We simply repeat the same process with each predicted words until we get a `<end>` token.

Unfortunately, the sampling process didn't lead to meaningful captions. This may be due to a bug in the implementation or due to lower training time of the models. Another reason that could lead to such problem is that the models are not properly over-parameterized to over-fit the dataset. In addition to that several other sampling technique such as Beam Serach [6] as suggested in literature could be used for experimentation. Another possible solution to the problem might be use the entire dataset so that each image has large number of captions associated with them.

4 Conclusion

In this project, we implement a method that combines ideas from both computer vision and natural language processing. We implement an encoder-decoder method to generate captions for given meme template. The encoder uses a CNN and a linear layer to generate image embedding. This embedding is passed to the decoder. The decoder uses the image embedding to obtain context variable and hidden variable. Then the decoder feeds embedding of each word in the captions through the LSTM. The sampling technique to obtain a caption is also similar except the first word fed into LSTM is the token `<start>`.

Acknowledgment

I would like to thank Mohammad Abubakar for the project idea. **Note:** Apart from the project idea, all the implementations and report were produced individually.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[2] Lawrence Peirson. Memeproject. `https://github.com/alpv95/MemeProject`.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014.

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society, 2016.

[5] Abel L. Peirson V and E. Meltem Tolunay. Dank learning: Generating memes using deep neural networks. *CoRR*, abs/1806.04510, 2018.

[6] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3156–3164. IEEE Computer Society, 2015.