MemeCaptcha: Generating Meme Captions Using Neural Networks

Muhammed Ali Binici
Department of Computer Science
University of Rochester
mbinici@u.rochester.edu

Thanatcha Khunkhet
Department of Data Science
University of Rochester
tkhunkhe@u.rochester.edu

J. Hassler Thurston
Department of Computer Science
University of Rochester
jthurst3@u.rochester.edu

Abstract

We compare the effect that different neural network structures have on the relatively original problem of generating captions for memes. Given an image, we would like to generate English top and bottom text for this image that would pass as a caption for a human-generated meme. While this problem is related to the image captioning problem, it is inherently different in a few key respects. First, the captions themselves that would make good memes are inherently different. Second, generating memes is an inherently creative task, and as such, there is no single "correct" caption or type of caption that one may output for an image that would make for a good meme. Even with these inherent difficulties, however, we are able to develop several models which are able to generate such memes.

1 Introduction

Within the past few years, there has been a lot of groundbreaking work on image and video captioning. Much of the work for image and video captioning involves using attention models [4], extracting features from videos and using background linguistic knowledge [8], and using LSTMs [9]. Whereas these papers aim to output a caption that accurately depicts what is going on in an image or video, our task is relatively more difficult, as these captions do not make good captions for memes. The Oxford-English Dictionary defines a meme as "an image, video, piece of text, etc., typically humorous in nature, that is copied and spread rapidly by Internet users, often with slight variations." [5]. For our purposes, we consider a more narrow definition of Internet meme, that is, an image together with English top text and English bottom text. Figure 2 contains an example of this more specific type of meme.

We then formalize the meme captioning problem as follows: Given an image, generate English top text and bottom text for this image, which would pass as a human-generated Internet meme. As can be seen in Figure 2, generating a meme caption is inherently different from generating an image caption. For this figure, a suitable image caption might be "A cat is wearing a knitted hat." However, this caption would pass as a rather boring meme. Instead, Internet memes often draw on more contextual elements of the image, including the specific entities in the image, and what they may be feeling.

Memes are also often created with a specific external context in mind. We suspect the meme in Figure 2 may have been created around Christmas time, though the image could have completely different top and bottom text that would make sense in another context. As such, unlike the image captioning problem, a given image may not only have multiple different *good* meme captions, but

it may also have multiple different *types* of meme captions, with each caption having a completely different semantic meaning.

As such, evaluating the accuracy of a computer-generated meme poses a unique challenge. We attempt to solve this problem indirectly, evaluating the performance of a given computer-generated caption by comparing the similarity to a human-generated caption for the same image. We evaluate and compare the effect that different similarity metrics have on performance, as well as the effect of using different network architectures. In the next sections, we summarize related work, outline our models, evaluate our results, and conclude by outlining potential future directions we think one must take when developing models for this relatively new problem.

2 Related Work

While the image captioning problem has been well studied, we only found two groups that have attempted the meme captioning problem. Wang and Wen [11] combine a reverse Google image search for keyword extraction with a nonparanormal model in order to generate captions for memes. Introduced by Liu et. al. [3], the nonparanormal model transforms an n-dimensional vector of an unknown distribution into an n-dimensional vector with a multinormal distribution. Wang and Wen use this approach as a means of encoding a sparse set of textual and visual features of memes into a form which is better suited for training.

We also found work by Zhang, Shao, and Liu [12] as part of an informal weekend competition [1]. The authors use the pre-trained Tensorflow im2txt model [10] for generating image captions, and plug in their own data set of memes downloaded from the internet. Unfortunately the authors do not supply more specific information about their training procedure, and do not report on the performance of their results.

Although Wang and Wen's model [11] outperformed a vanilla RNN that used no features from the images, the authors failed to compare the effect of using deep recurrent neural networks to generate captions, after performing feature extraction from the images. Wang and Wen also make heavy use of the Reverse Google Image Search feature, as well as their own TF-IDF [2] meme search engine, to query existing meme captions which match a particular word or phrase. After investigating their methodology further, we found that in order to replicate their model, we would need to automatically perform reverse Google image searches on the fly, at a relatively high rate, without Google detecting us as a robot. As such, we were unable to replicate Wang and Wen's nonparanormal model.

Instead however, we note that many of our models combine the two approaches in [11] and [12]. Most of our models are based off of Tensorflow's *Show and Tell* model [10], where we encode an image into a fixed-length vector using a convolutional neural network, and then decode the fixed-length vector into captions using a recurrent neural network. Since each element of the fixed-length vectors outputed by our encoder are already distributed non-sparsely between 0 and 1, there is no need to apply a probability integral transform to these features to make them uniform, or multinormal, before sending the vector to a decoder. As such, the need for the nonparanormal becomes somewhat irrelevant.

Following existing work by [12], we approached our goal by modifying the already existing image captioning method developed by Google [10]. This *Show and Tell* model (Figure 1) is an encoder-decoder model that uses inception v3 architecture [7] for the encoder and long short-term memory (LSTM) for the decoder. The inception v3 architecture is a deep convolution neural network (CNN) which is widely used for image tasks and is a state-of-the-art model for object recognition and detection.

The encoder produces a representation of an image input by embedding it to a fixed-length bit vector, which is then used by the recurrent neural network (RNN) decoder as an initial hidden layer to generate sentences. To make the image embedding vectors usable by the RNN, a single trainable layer is added on top of the Inception v3 model to transform the image embedding into the word embedding vector space. The decoder is an LSTM, or a recurrent neural network that has point-wise operations and can hold "memory" longer than a simple RNN. The ability to hold memory for a longer time makes LSTMs very fit for sequence modeling tasks such as language modeling and

[1] CalHacks 3.0 [2], a 36-hour hackathon for undergraduate students

[2] Term Frequency, Inverse Document Frequency

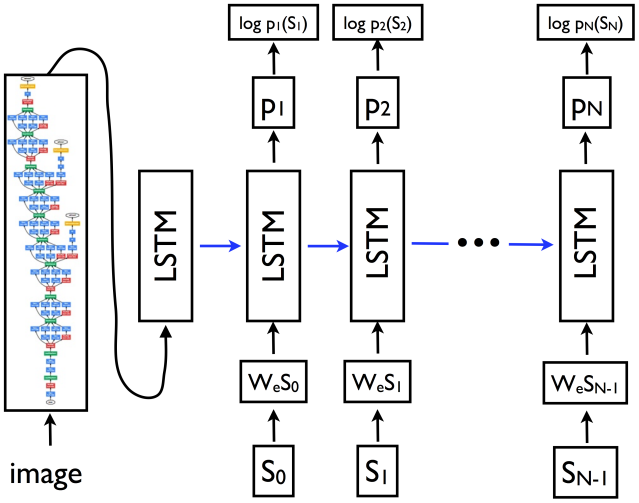

Figure 1: Google's Image-to-text Architecture. Picture taken from [10].

machine translation. In the *Show and Tell* model, the LSTM network is trained as a language model conditioned on the image embedding.

At a given time t, the LSTM is fed as input a word S_t. This word is represented as a bit vector, with a 1 in position i in the bit vector if the bit vector encodes word i, and a 0 everywhere else. In the diagram of *Show and Tell* model (Figure 1), $S_0, S_1, .., S_{N-1}$ are words of the caption and $W_e S_0, W_e S_1, .., W_e S_{N-1}$ are their corresponding word embedding vectors. The network outputs probability distributions for the next words in the sentence $p_1, p_2, ..., p_N$. The log-likelihoods of the correct word at each step are $logp_1(S_1), logp_1(S_2), ..., logp_N(S_N)$. During training, the *Show and Tell* model compares the log probabilities of the output word to the next input word, and computes a loss function from that.

Figure 2: A sample meme: an image with top text and bottom text.

3 Methods

3.1 Data Collection and Preparation

Inspired by Zhang et. al. [12], who wrote Python code to scrape the Internet for memes, we implemented and ran a distributed scrape job on the `csug.rochester.edu` network using Scrapy [6]. Over the course of two days, we were able to download millions of memes from MemeGenerator[3], a website in which any Internet user can create memes with top and bottom text. Each downloaded meme contains JSON which provides us with a link to the original image used, as well as the top and bottom text, of the human-generated meme. The top and bottom text, which we collectively define as a meme caption, will serve as the "ground truth" for training the network. We were thus able to avoid performing optical character recognition to extract the top and bottom text from an image.

Since we are only considering English captions, and since users may generate memes in any language, we made use of a simple heuristic to rid our data set of most non-English memes. If the top text or bottom text contains any non-ASCII characters, we labeled the meme as not English, and discarded the meme from our downloaded set. Otherwise, we assume the meme is English. From an empirical analysis of our resulting dataset, this simple heuristic performed extremely well, but was not perfect. For example, a Spanish meme without accent marks over characters will be labeled as an English meme, and an English meme with a smiley-face emoji would be labeled as non-English.

The above preprocessing step eliminated approximately 70% of our downloaded dataset. However, we were still left with 16,409,055 memes, a very sizable amount for training. From these approximately 16.4 million memes, 22,349 unique images were used. As only the images will be used as input during training, we randomly partitioned the images into a training set and a validation set (used for testing), to preserve a ratio of training-test of approximately 2:1 [4]. To do this, we shuffled the images and randomly assigned 66% (14,750) of the images to be in the training set, and the rest to be in the testing set.

From another empirical analysis of our data set, some images were used frequently when generating memes, and other images were used rarely. To be consistent with the *Show and Tell* model, to reduce the size of training, and to avoid overfitting, we decided to pair each image with only the top five meme captions for that image on MemeGenerator. The top five captions were selected based on the number of upvotes this meme received on MemeGenerator, with ties broken arbitrarily [5].

3.2 Models Architectures, Trainings, and Evaluations

We implemented and examined the performance of two different neural network models listed in Table 1 below. Models #1 and #2 are modified from TensorFlow's *Show and Tell* image-to-text model, and the other models, which we did not get decent results for but have implemented, are also similar.

3.2.1 Inception V3 + LSTM

As described above, the *Show and Tell* model consists of an encoder, which is a deep convolutional neural network that encodes an image into a fixed-length image embedding vector, and a decoder, which is an LSTM that decodes a fixed-length word embedding vector and produces a caption. In between the encoder and decoder, there is a unit that translates the fixed-length image embedding vector in the image embedding space, to a fixed-length word embedding vector in the word embedding space. This word embedding vector will change through time, and represent the hidden state of the LSTM.

Training the *Show and Tell* model comprises two phases. During the first training phase, the parameters in the LSTM network are trained with respect to the word embedding output from the added top layer of the Inception v3 network, while the parameters in the Inception v3 network is kept

[3] `http://memegenerator.net`

[4] The validation set was only used to test the performance of our models, and not to pick the optimum parameters of a given type of model. As such, our *validation* set was semantically equivalent, and used as, a testing set. The terms *validation* and *testing* are overloaded here to reflect the *Show and Tell* model's terminology, as they used the validation set truly in a validation sense, i.e. for picking the best model to submit to a competition.

[5] If an image was used less than five times, we output less than five image-caption pairs.

Table 1: Models

Model Number	Structure	Input	Output
1	Inception V3 + LSTM	meme images	meme captions **
2	CNN + LSTM	meme images	meme captions
3	LSTM***	image captions*	meme captions

* output from the pretrained *Show and Tell* model with ILSVRC dataset to generate image captions
** both top and bottom texts *** results not reported in depth

fixed. Then, in the second training phase, all parameters (including the parameters of Inception v3) are trained to jointly fine-tune the image encoder and the LSTM.

3.2.2 CNN + LSTM

Training the inception modules of model #1 take a significant amount of time and resources, and thus we would like to see if a simpler model could achieve the same performance. Instead of using the entire inception v3 model like the encoder part of the *Show and Tell* model, the CNN + LSTM model uses only the first convolution layer of Inception v3. That is, out of the many layers of the Inception v3 encoder, we take the output of the first layer and treat it as our image embedding vector, which is then translated to a fixed-length word embedding vector to be fed directly into the LSTM decoder. The model was then trained – through both training phases like in the model#1 training. During the first stage of training for this model, the initial weights for the first layer of the Inception v3 module were used.

3.2.3 LSTM

Nonetheless, training the CNN could take a long time still and we already have a pre-trained model for generating regular image captions. Using the image captions the *Show and Tell* model generated, we trained 2 sequence-to-sequence models. For these models, we had 22k images at the beginning each having 3 different image captions and many more meme captions. From this data we created a dataset of 86500 pairs of image and meme captions. After dividing the dataset into training and validation sets with ratios 1:1 and 3:1, we trained both of our models on these sets.

The first model which is called seq2seq is powered by Tensorflow [1]. It consists of 3 LSTM layers with 1024 units on each and uses the Softmax loss function. The second model which is called Opennmst neural machine translator is powered by Torch and is also constructed using LSTM cells.

Since the vocabulary files for image captions and meme captions were disproportional, 750 vs 51000, it was hard to get the perplexity down to 1 digit. The seq2seq model started with learning rate 0.5000, step-time 2.05 and perplexity 910.24 and after 6400 steps, we got perplexity 34.53. Although the trained seq2seq model do not create funny and meaningful meme captions for most of the test images, it still learns enough from the training set to produce captions like "I ' m gonna be you know you".

On the other hand, the Opennmt model, after running 7 epochs, was able to bring perplexity from 8370.67 to 380.71. Again, because of the disproportional vocab files, it was hard to optimize the model. When we tested the trained Opennmt model, it only produced unknown tokens ("<unk> <BC> <unk>").

As a future work, we can collect more memes to create more proportional vocab files for train and dev sets.

3.3 Experimental Setup and Performance Comparison

The models were trained using the training data as described in the previous section. In addition, Models #1 and #2 each were trained with two different dictionaries, corresponding to two different minimum word count thresholds – 1 and 4. The minimum word count threshold determines the number of occurrences of a word to allow the word to be included in the dictionary. With a threshold of 1, every word is included in our dictionary, even if it only appears once in one training set meme.

With a threshold of 4, a word must appear at least four times to be included in the vocabulary - words that appear fewer than four times are marked with a special unknown token.

In order to make the models comparable, Models #1 and #2 were trained using the same data, for the same amount of training time (10 hours for each training phase), and with the same computational resources (server: `bh25fen.circ.rochester.edu`, number of GPU nodes: 2, memory per GPU: 24 GB).

4 Experiments

4.1 Loss vs. Iterations

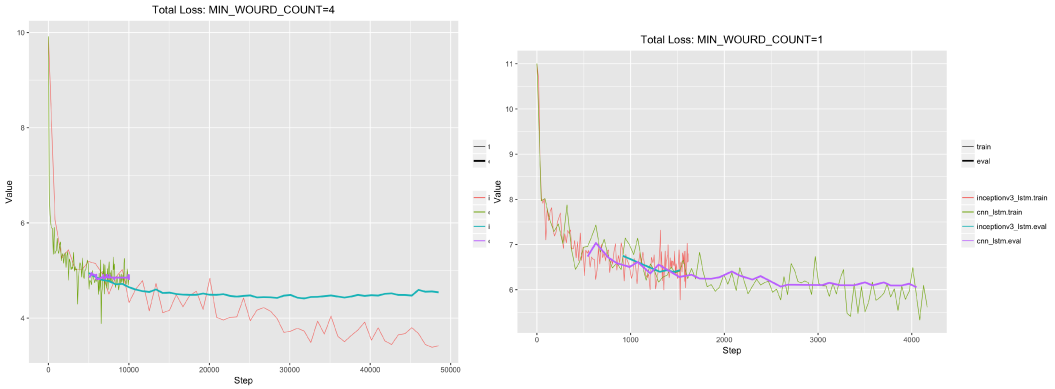

Figure 3: Total loss vs. number of iterations. Left: Models for which the threshold for minimum number of times a word appears = 4. Right: when threshold is 1.

Figure 3 plots the loss of each of our LSTM models vs. number of iterations. As can be seen, for each model, the loss function decreases sharply during the first few iterations, and then either continues to gradually decrease or holds steady as the number of iterations increases. Please note the difference in scale for the x-axes for the left and right plots. This corresponds to how many iterations were performed for each of the models during the same amount of training time. We see that the plots on the left train for a shorter amount of time per iteration, because the size of the vocabulary is smaller.

Interestingly, the Inception V3 + LSTM model with a threshold of 4 seems to start overfitting at around 10,000 iterations of training. That is, the loss on the validation set starts to be significantly higher than the loss on the training set. As we will see later, this does not correspond to the empirical performance we obtained in Section 4.3.

4.2 Perplexity vs. Iterations

Figure 4 plots the perplexity of each of our LSTM models vs. number of iterations. As can be seen, in general the perplexity decreases sharply at first, and then holds steady as the number of iterations increases. Unfortunately we were not able to obtain perplexity results for the Inception V3 models for a low number of iterations. We must therefore infer that the perplexity of these models for low iteration counts starts high and decreases rapidly.

This makes sense intuitively, as perplexity is implicitly a measure of the confidence that the model has in generating the next word correctly. However, while the obtained perplexity graphs show that training is performing reasonably well, we reason that the perplexity metric may not be the best

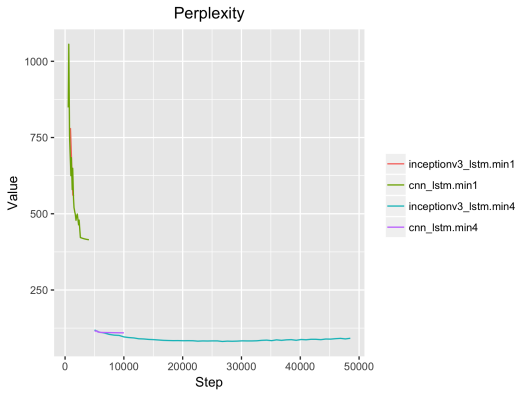

Figure 4: Perplexity vs. number of iterations

metric to judge the performance of a model for the meme captioning problem, as discussed in a later section.

4.3 Generated Output

Figure 5 contains sample memes outputed by our models on an image in our validation set. As can be seen, we obtain the best results when we set the minimum word count threshold to 4. Interestingly, setting the minimum word count threshold to 1 produces words which are very common in our data set, which we did not expect to find. Instead, we expected that adding more words to our vocabulary would increase the likelihood that obscure words would appear in the captions. In addition, the models with a threshold of 1 tend to produce captions of shorter length than the models where the word count threshold is 4. We do not have an explanation for these phenomena, other than the fact that this model was only able to go through 1600 iterations, in the same amount of time as the Threshold 4 model was able to go through many more iterations.

Figure 5: Output captions of our models for an image in our validation set. Left: Inception V3 + LSTM model, word count threshold = 4. Center left: Inception V3 + LSTM model, word count threshold = 1. Center right: CNN + LSTM model, word count threshold = 4. Right: CNN + LSTM model, word count threshold = 1.

4.4 Convergence

Training our models for a longer amount of time may help to mitigate the above problems. However, we do not suspect the problem would be solved entirely by a longer training cycle. Figure 6 shows an example of a training set image from our data set, and the result of running the Inception V3 + LSTM model with threshold = 4 on this image. The result is a rather strange output, which can only be explained by looking at our data. In our data set, we found numerous examples of this image with words "Erik Kolacek" followed by rude or inappropriate remarks about Erik Kolacek. We suspect that a user of MemeGenerator did not like a colleague named Erik Kolacek, so went on MemeGenerator and created a plethora of memes about Mr. Kolacek to vent his/her frustration. As such, one of our models has learned to output a similar caption. This is not so much a problem with our model, as it is a problem with the data set we used.

Figure 6: Output of the Inception V3 + LSTM model on the above training set image. We believe the image was used primarily to vent frustration at Erik Kolacek, and as such, we conclude that arbitrary memes from MemeGenerator may not make for the best training data.

Our original purpose in using data sets from MemeGenerator was that we were able to obtain millions of image-caption pairs, and furthermore these pairs would function as a "gold standard", since they were generated by real humans. We now realize that just because memes were generated by humans does not imply that these memes should live up to such a gold standard. We tried to get around this problem by only including the top 5 captions for each image when training, though unfortunately this was not enough. As such, future researchers must carefully consider the question of which meme data set to use for training.

5 Conclusion

Our MemeCaptcha give a preliminary framework for anyone who wants to study generations of meme captions using deep neural networks. Among the three network architectures studies, we conclude that the Inception v3 + LSTM model with minimal word count threshold of 4 perform the best empirically. Nonetheless, future work is necessary to improve the model performance. The Possible future work includes cleaning the dataset, obtaining a better dataset from other meme sources, and comparing the results using different metrics than loss and perplexity. We look forward to the continuing work that may be done on this interesting and original problem.

References

[1] Denny Britz et al. "Massive Exploration of Neural Machine Translation Architectures". In: Mar. 2017. arXiv: 1703.03906 [cs.CL].

[2] *Cal Hacks 3.0 | Devpost*. Accessed: 2017-05-15. URL: https://calhacks3.devpost.com/.

[3] Han Liu, John Lafferty, and Larry Wasserman. "The nonparanormal: Semiparametric estimation of high dimensional undirected graphs". In: *Journal of Machine Learning Research* 10.Oct (2009), pp. 2295–2328.

[4] Xiang Long, Chuang Gan, and Gerard de Melo. "Video Captioning with Multi-Faceted Attention". In: *arXiv preprint arXiv:1612.00234* (2016).

[5] *Oxford English Dictionary Online*. http://www.oed.com/. Mar. 2017.

[6] *Scrapy | A Fast and Powerful Scraping and Web Crawling Framework*. Accessed: 2017-05-15. URL: https://scrapy.org/.

[7] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[8] Subhashini Venugopalan et al. "Improving lstm-based video description with linguistic knowledge mined from text". In: *arXiv preprint arXiv:1604.01729* (2016).

[9] Subhashini Venugopalan et al. "Translating videos to natural language using deep recurrent neural networks". In: *arXiv preprint arXiv:1412.4729* (2014).

[10] Oriol Vinyals et al. "Show and tell: Lessons learned from the 2015 mscoco image captioning challenge". In: IEEE, 2016.

[11] William Yang Wang and Miaomiao Wen. "I Can Has Cheezburger? A Nonparanormal Approach to Combining Textual and Visual Information for Predicting and Generating Popular Meme Descriptions." In: *HLT-NAACL*. 2015, pp. 355–365.

[12] Andy Zhang, Andrew Shao, and YuXuan Liu. *Dank Meme Generator*. Nov. 2016. URL: https://devpost.com/software/dank-meme-generator-rk4o31.