Report

Jia Sheng and Fanyi Meng
Neural Pragmatic NLG WS 22

Introduction

This is the report for the final project of the course 'Neural Pragmatic NLG WS 22', where we re-produce a paper. We choose *Pragmatically Informative Image Captioning with Character-Level Inference* (Reuben Cohn-Gordon, Noah Goodman, Christopher Potts, 2018), where a neural image captioner is combined with a Rational Speech Acts (RSA) model to produce captions that are both true and distinguishable among inputs of similar images.

The authors train some production and evaluation models with a CNN-RNN architecture on Character-level and word-level respectively, using image and caption data from MSCOCO (henceforth coco) and Visual Genome (henceforth vg). One production model produces caption after multiple images with similar content are put into the RSA model, with output from two speakers: literal and pragmatic speakers, describing the image at index 0. The evaluation model is used to test upon leaning the captions from the two speakers, which caption has a higher probability to let the hearer point to the target at index 0.

The evaluation data involves groups of 10 images with common objects such as tree, bus, and person, taken from vg dataset. The result shows that the pragmatic speakers obtains higher accuracy than the literal speakers. While the word-level literal speaker performs better than the char-level literal speaker, the char-level pragmatic speaker out performs the word-level pragmatic speaker.

Original Code and Models

The paper provides a link to the GitHub repository for the codes used in this paper, which is later cloned to our project repository. After solving some errors (mostly due to library deprecation and wiki image limiting the accessibility to prevent web-crawlers, or so we assume), the main programme is runable with variable results. Additionally, we notice that in multiple *.py* files, multiple variables are used without initiation, and some file path leads to non-existing folders. The code provided is not complete.

The pre-trained models provided by the authors include:

- *coco-encoder-5-3000.pkl* and *coco-decoder-5-3000.pkl*, presumably encoder and decoder trained on coco dataset;
- *vg-encoder-5-3000.pkl* and *vg-decoder-5-3000.pkl*, presumably encoder and decoder trained on vg dataset;
- *lang_mod-encoder-5-3000.pkl* and *lang_mod-decoder-5-3000.pkl*, which is not mentioned in the paper, and upon finishing this report, we still do not know the purpose of them (therefore we ignore them in our re-production).

Some of the main issues include:

- Test sets are not provided, and there is no further information on how to gather 'regions in Visual Genome images whose ground truth captions have high word overlap' for TS2;
- Word-level models are not provided, and the existing char-level might not work with a word level dictionary (this has been proved to be the case with our *seg_type_word* branch).

Proposed solutions are:

- Incomplete code: we will try to rummage through what is provided, and add component with copy-and-paste programming;
- Test sets: only one test set is used as opposed to the original proposal, which is test set 1 with '100 clusters of images, 10 for each of the 10 most common objects in Visual Genome';
- Missing word-level models: we have to train our own. This leads to another problem that there is no specifications of model training in the paper, therefore we have an enormous space from the authors word-models. A better solution would be to train our own char- and word-level models. But then we face the risk that our models would not work (or as well as those of the authors), and will fail to test anything.

Word Model Training

Data Preparation

Before the vg website is down, we fortunately have downloaded some images and *region_descriptions.json* already (We choose this file according to the code from *utils* folder from the authors). We verify that we can concatenate the information with a url base to get the working url of the images, so that we still get access to the images without downloading them; and that they are of the same images as those we have managed to download.

We are not able to train our image encoders for many reason, so we use the encoder provided by the authors. Therefore the number of features also follows (1, 256). The length of captions are set to 10 words (plus the start and end symbols). A vocabulary class is imported from the original code from the authors. We pre-process all the captions in the vg json files, and only count those words that have appeared at least 100 times, yielding a vocab size of 4,987. Since most of vg regions are small, the process is relatively fast on Google colab with GPU.

The result is the pre-process halts at 40k data points due to the mono channel of grey-scale images causing error during the process. But 40k training data seems to be enough for our small model (so we guess). They are saved as two-tuples containing a tensor of image features (forwarded by the pre-trained CNN model), and a tensor of captions as index basing on the vg-dictionary. The test data contains 10 groups of data points, each contain 4k four-tuples (two aforementioned tensors, a string of the ground truth caption, and a string of url of the image).

We prepare the coco data in the same way. The problem we found later is that most of the coco captions are longer than 10 words, since they describe a complete image,

contrasting a regional description from a vg data point. And also due to the size of the images, the process is longer. In order not to get suspended for abusing Colab's GPU, we save the processed training data into 10ks, and plan to initiate 4 dataloader during the training process (But we still manage to get connection errors running the programme on Colab and Urobe).

Code completion

Apart from adapting code for processing, the biggest challenge is to make the training loop run (for we might get the dimensions wrong). So after several round of debugging the authors code, and some exercise sheet from the course (for the training, mainly from sheet 5.1, and Sheet 8.1 for sampling), we managed to get the training run (and later with some trained decoder, we re-wrote the sampling function).
But due to the many references we take, there are some differences (and other that we aren't aware of):

- At one point, the author used one-hot encoding of the shape (1, 1, 30) for chars upon initiating a *state* class object, and saved the encoding into a class variable. Then before this value could be used anywhere, it's assigned a new value *[''']*. We therefore ignored one-hot encoding;
- The original shape used by the authors is of three dimensions, where we only use two;
- We later found that during unrolling, the original models seem to take the image features first and then one char (embedded) at a time. But since we do not see the code for the training models, we instead concatenate the image feature with each word embedding, and feed the tensors into the model, ending up with a different shape for lstm.
- Sample is therefore also changed due to the different output shape (but even if the shape is not changed, we will still have to make big change since we can't see what the original code is trying to achieve).

Training

We initially train our models on Google Colab (because of their GPU), then the constant time-out and suspensions do not allow us to finish the training there, therefore we borrow some additional space on Urobe and move the training there. But still, Colab is very useful when we test out a few epochs' training to find out the best learning rate. Some examples for vg word models are (from the project repository):

epochs	lr=0.005	lr=0.001	lr=0.0005
5	4,937,966	3,526,568	3,141,721
10	5,115,217	2,794,233	2,046,428
15	5,394,896	2,349,121	1,557,936

Later we found that a screen on Urobe would require a renewed Kerberos ticket every 24 hours, which forbid a very long training session. Additionally at one point, the loss stops converging but the number is still huge. As a result we train our models for certain epochs, save the model, generate a new key, resume training (and reduce the learning rate further if necessary).

We end the training for vg model after 105 epochs, and save multiple models in case of over-fitting. We do the same for the coco word model. The contrast in loss is obvious: vg model converges well since the image is small, and there is a one-to-one relation. The loss reduces from the original 738,076 to 4,649 (0.63% of the original epoch) in the last epoch. Meanwhile each coco image could point to several different captions, and that could have led to more uncertainty. The loss reduction is therefore only from 1,297,341 at epoch 1, and 123,273 at the last epoch 116, still 9.5% of the initial epoch.

The vg word model works well on training data, but not so good on testing data. The generations are mostly grammatical. But the data the model is trained on contain normally very simple object such as a stapler, a clock, etc. If presented an item not presented in the data, the model will not recognise it, and predict something interesting such as *red car leaning on pole* (the target *red and blue banner on side of building*). In contrast the coco model some times generate ungrammatical or/and incomplete captions, due to us setting the max sentence length too small. And on multiple occasions, the model predicts dogs when there aren't any. Therefore, there could be some biased training data.

Word-level RSA model

Since as previously mentioned, the output shape of the word model is different from that of char model, in order for us to use the same unrolling and to adapt to the original RSA model, we need to modify multiple files. At this point the word model is separated from the main branch (which has the original char model).

We use greedy search and apart from predicting things that are not in the images, most of the time the captions generated by literal and pragmatic speaker are the same, but with slightly different log probability.

Evaluation

Data Preparation

According to author's paper, they evaluated the performance of pragmatic speaker (s1) in comparison to literal speaker (s0) with two test sets, each consisting of 100 clusters, in which each cluster again consists of 10 images that have a similar object. For Test set 1, the author selected 10 most common objects in Visual Genome, namely, man, person, woman, building, sign, table, bus, window, sky, and tree. For Test set 2, the author took 100 clusters of 10 images whose ground truth captions have high word overlap in Visual Genome.

As the author did not provide his test sets 1 nor 2, in build_test_data.py we went through the VG dataset and, based on the ground truth captions of regions in each image, extracted the first 100 entries describing each one of the 10 most common object. Then, we cropped the image to get the exact regions of these 1000 entries. These 1000 image regions build up our Test set 1. As it is unclear how the author defines "high word overlap" and thus the resulting difficulty in reproduction, we did not reproduce Test set 2 anymore - but it is expected that it will yield the same result.

Caption Generation

Having built Test set 1, the next step is to generate the S0 and S1 captions of these 1000 images. generate_captions.py adopts author's coco-trained char-level CNN-RNN model as well as RSA model, and generates a literal (S0) caption and a pragmatic (S1) caption for the target image given each cluster (we took the first image in every cluster as the target image, and the other 9 images serve as distractors). We use the same hyperparameters as in the paper, which is a beam width of 10 and a rationality of 5.0 for S1.

Similarly, generate_captions_word.py generates S0 and S1 captions for images in test set 1, but using our own coco-trained word-level RNN decoder. Also, due to the large vocabulary of our word-level model (4533 words), the beam search unrolling method is expected to take an eternity, so we only have greedy generated captions, just as in author's case.

Automatic Evalution

With the captions generated by coco-trained char- and word-level models, we then made use of the models trained on VG data to evalute the success of these captions. Like in author's paper, we define a caption as being successful, if the target image has the highest probability under the distribution $L_{eval}(image|caption)$ in comparison to the other 9 distractors in the same cluster.

In get_accuracy_ts1.py, the char-level models are evaluated. We achieved that by passing the s0 and s1 captions generated by char-level COCO models into char-level VG models:

1. First of all, each image is tranformed into a vector of features by the VG CNN encoder, and each coco-generated caption is transformed into a vector list of segments.
2. Then, for each image in the 10-images cluster, its image features are passed into the vg-decoder to generate the probability distribution of the first caption segment and the hidden layer output.
3. After that, we iterate over all segments in the coco-generated captions, and each time pass the hidden layer output from the last iteration together with the current coco-caption segment into the vg-decoder to obtain the probability distribution of the next segment. In this step, we record the predicted probability of the true next segment in the coco-caption.
4. Finally, after iterating over all segments of the generated caption, the overall probability $P(image|caption)$ is calculated by adding up each vg-predicted coco-segment probabilities.

After iterating over every image in the 10-image cluster, we obtained a probability $P(image|caption)$ for every image and every caption. Then, for each caption, we checked whether the probability of the target image given this caption is the highest among the 10 images cluster. If $P(image|caption)$ is the highest, we recorded that this caption is successful in capturing the discriminative features of this image, and otherwise failure.

The number of total successes are used for evaluating the model-generated captions.

In get_accuracy_ts1_word.py, we repeated the same evaluation process, but this time for the word-level model-generated captions. Due to the different decoder structure, in each forward pass we passed not only the caption segment, but also the image features into the word-level decoder. Then we again calculated the overall probability $P(image|caption)$ and recorded the number of successes.

Outcome

We recorded the detailed evaluation outcome from automatic evaluation in get_accuracy_ts1_output.txt (for char-level models) and get_accuracy_ts1_word_output.txt. Below is a summary:

	Char S0	Char S1	Word S0	Word S1
greedy	43	44	6	7
beam	38	49		

Table 1: Our TS1 evaluation results

Then, we compared it to the author's evaluation result:

	Char S0	Char S1	Word S0	Word S1
greedy			57.6	60.6
beam	48.9	68.0		

Table 2: Author's TS1 evaluation results

Putting 2 tables together, we have the following observations:

- For char-level models, despite that we had to build our own Test set 1 (which the author did not provide), our evaluation using beam search unfolding correctly replicates the author's results, and confirms that the pragmatic speaker (S1) performs better at producing captions that distinguish images from their similar kind than literal speaker (S0).
- For word-level models, as we had to train our own decoders for COCO and VG dataset (which the author did not provide either), and considering the limitation of our training capabilities, we believe the comparison with author's results is no longer meaningful here. Also, although word-level S1 is slightly higher than word-level S0, this difference might be led by chance and cannot be used as proof for S1 outperforming S0.
- Surprisingly, when evaluating the char-level models with captions generated by greedy unfolding, there is no signicant difference in the accuracy score of S0 and S1. In this case, although we cannot directly compare our accuracy scores of char-level and word-level models, we can put the success of character-level model in producing more distinguishing captions with RSA model in question. A more fair evaluation could only be acheived if the author could provide the accuracy scores of his char-level models with greedy unfolding as well.

Conclusion

In this reproduction project, we trained our own word-level models, generated the test data, and carried out the evaluation. Besides the author's original setup, we also tested the character-level models using greedy unfolding, the result of which confirms the drawback in the authors' approach – that their conclusion of the "superiority of character-level model" could only be confirmed if they evaluate not only word-level models, but also their character-level models with greedy unrolling.