

Improved OBJ2TEXT

Wen Xiao

Department of Computer Science
University of British Columbia
xiaowen3@cs.ubc.ca

Ke Ma

Department of Computer Science
University of British Columbia
polyumark@gmail.com

Sakura Tamaki

White Graduation Society
Shirasaki Academy
tamaki.sakura@hotmail.com

Abstract

Generating visually descriptive languages from object information serves as an intermediate step in image captioning. While the prevailing object-to-text models, including OBJ2TEXT, can produce plausible visual descriptions, they are based on conventional recurrent neural network architectures and suffer from long-range dependency issues. To address this problem, we proposed two variants of OBJ2TEXT model: 1) Google Transformer based model, and 2) Pointer Network based model. Relying entirely on the attention mechanism without any recurrence and convolutions, Google Transformer based model is able to provide direct linking between distant positions. Pointer Network instead is a sequence-to-sequence model with an additional attention module allowing explicit extraction of words from encoder to output. Experiments are conducted on the MS-COCO dataset using BLEU, METEOR, ROUGE-L and CIDEr score. The results show that Google Transformer based model outperforms the baseline model in all evaluation metrics, whereas Pointer network based model better handles the count of objects.

1 Introduction

Image captioning has recently received considerable attention, but solving such problem is still hard. Instead of direct training from real image, Yin and Ordonez (?) proposed a method called OBJ2TEXT. This method generate image captions base on the object layout of the image only, where the object layout consists of a set of objects and their locations. However, this model suffers from numerous issues. In our project, we are aiming at proposing variants on their model to generate better visually descriptive language from object layout. The two main issues in the original model we are aiming at resolving are 1) adapting a better

way to incorporate visual features in the context of generating captions from object layout instead of just using output from VGG16 (?) and 2) reducing or even eliminating long-range dependencies in the network using other sequence-to-sequence model instead of LSTM.

2 Related Work

OBJ2TEXT. The basic OBJ2TEXT (?) is a sequence-to-sequence encoder-decoder type of model. Its encoder is a Long Short-Term Memory (LSTM) network that combines word and object layout for multiple objects into a fixed length vector. Consequently, it encodes the spatial relationships between objects. Then, this vector is passed to a LSTM decoder for caption generation. One variant of this, namely OBJ2TEXT-YOLO, incorporates the YOLO object detector (?) to enable caption from real image to text. Another variant, OBJ2TEXT-YOLO+CNN/RNN, adds VGG16 (?) to encode the image, enabling caption generation from both visual feature and object layout encoding.

Sequence to Sequence Model. Recently, numerous improvements had been made on Sequence to Sequence model, mostly aiming at solving LSTM's problem on capturing long term dependencies. Vaswani et al. (?) proposed what is commonly called as "Google Transformer Model" where a sequential model might not be necessarily needed and one could use self-attention to achieve even better performance in a sequence-to-sequence model. (?) proposed a method to rapidly access element from the previous sequence using distributed attention, which is called "pointer network".

3 Our Approach

To reduce the distance in text generation, we propose two solutions.

3.1 Google Transformer

The first model we incorporated into the OBJ2TEXT framework is the Google Transformer model (?). Unlike other conventional seq2seq frameworks that use recurrent neural network architecture, Google Transformer model applies a self-attention mechanism to directly model relationships between all words in a sentence, regardless of their respective position. In this way, temporal information is connected without the use of any complex recurrent or convolutional neural networks. In fact, since the inputs to OBJ2TEXT are only the object’s category and its layout information (i.e. location, weight and height of the bounding box) irregardless of the sequence order passed to the encoder, it exactly matches the specification of Google Transformer model because it requires no particular order information. Hence, Google Transformer is an ideal model in the context of OBJ2TEXT.

It is noteworthy that Google Transformer addresses the difficulty of learning long-range dependencies between distant positions, which has long been a key challenge found in many sequence transduction tasks using traditional recurrent neural networks. Acting as a quick link between different positions in a sentence, the self-attention mechanism captures all necessary temporal information in a constant number of operations, which yields faster and better performance on machine translation tasks.

The structure of our Google Transformer based model is shown in Figure 1. The left part of the figure denotes the encoder part and the right part of the figure denotes the decoder part. All

dashed lines represent how the attention is combined. More specifically, the self-attention layer at the bottom brings sequence related information into the current position. Then, the encoder-decoder attention layer at the upper part attends to the output from the encoder and brings it to decoder. There exists a gap in the encoder-decoder attention layer since the outputs here are solely a linear combination of output from encoder. However it does not mean that the information learned at the lower layer of the decoder is not used. It is first used to calculate the attention and then it is used in the residual connection.

3.2 Pointer network

We used a modified pointer network (?) to generate the objects in the LSTM, this means that instead of remembering the exact word of the object type, the LSTM could choose only to remember where could it find the object type, and the generation of the name of that object will be the responsibility of pointer network instead of the sequence network, reducing the need to remember the exact word throughout the sequence. This idea inspired by the work of See et al.(?) and our model was slightly modified to deal with the problem of we might want to generate the same lemma as the word appears in encoder but in different form.

More specifically, let i to be the index for encoder, j be the index of decoder, o_i be the i -th object embedding and w_i be the word associate to this object, \hat{w}_j be the j -th word in the caption generation, h_j be the hidden layer of LSTM at time step j . Consider the probability of \hat{w}_j being generate when knowing the hidden state of h_j . In a LSTM decoder, we have:

$$P(\hat{w}_j|h_j, \{\hat{w}\}_{j-1}) = \text{softmax}(\text{LSTM}(h_j, \hat{w}_{j-1}))$$

In a pointer-generation network, we have:

$$\alpha_{ij} = \text{softmax}(h_j W_a o_i) \quad (1)$$

$$p_{ij} = \text{sigmoid}(W_o \times (\sum_i \alpha_{ij} o_i) + W_p \times h_j + B_p) \quad (2)$$

$$P(\hat{w}_j|h_j, \{w_i\}_i, \{\hat{w}\}_{j-1}) = \text{softmax}(p_{ij} \times \text{LSTM}(h_j, \hat{w}_{j-1}) + (1 - p_{ij}) \times (\sum_{i:L(w_i, \hat{w}_j)} h_j W_a o_i)) \quad (3)$$

where W_a is the weight of attention over the ob-

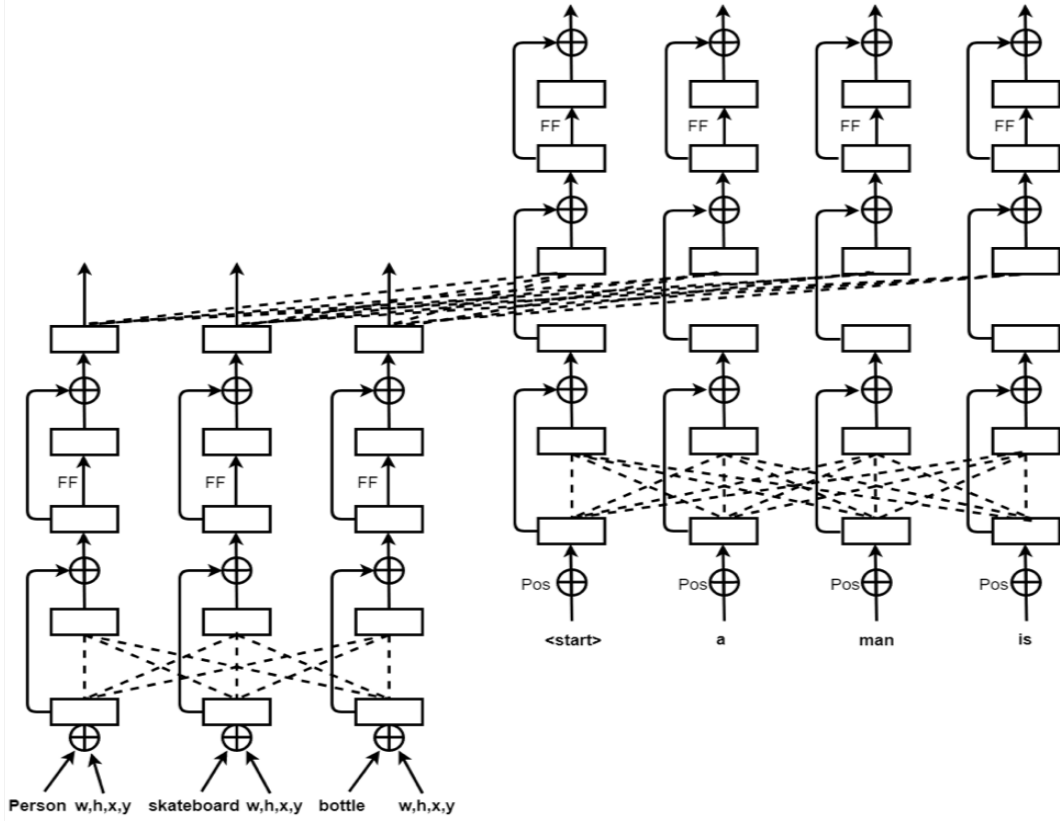


Figure 1: Google Transformer

jects, $L(w_i, \hat{w}_j)$ will return true if and only if w_i and \hat{w}_j has the same lemma, and p_{ij} is generated by a small pointer network, which determines how important the attention is in the current stage. If p_{ij} is large, it means we need to take more consideration for the objects in the encoder, the word we need to generate in the current stage should probably be a noun from one of the objects encoded.

To better balance the weights from the attention part and output of decoder, we clamp both values into range $[-100, 100]$.

The architecture is shown in Figure 2, we will take an example to explain how this works. First to note, we apply the same procedure for every word, it not only happens on noun. In this example, $p_{gen} = 0.7$, which means the 70% of the result come from the output of decoder and 30% of the result come from the output of attention module. And within the attention module, it is easy to find word 'glass' with highest weight, which means the word needed to be generated at the current step should be very related to the word 'glass', after combining the results and softmax function, the final word generated is 'glass'.

4 Experimental Results

Our experiments are conducted on the MS-COCO Validation Set (?). We didn't do it on the test set since the ground truth object layout is not provided on the test set. The hyperparameters are chosen on the Training Set to avoid overfitting.

4.1 Quantitative Results

We will show some quantitative results we have for the three models in this section. First, we use the official validation code to evaluate our results of the validation set (?), and the scores are presented in Table 1.

The performance seems close. Nonetheless, the Google Transformer based model outperforms the baseline with slightly higher scores for all evaluation metrics. The most significant improvement is observed in BLEU and CiDer scores, with an average increase of 0.1 for each BLEU score, and 0.2 for CIDEr score. For the Pointer Network based model, the performance is similar to the baseline.

in Figure 3, we show the histograms for BLEU-1, CIDEr, ROUGE-L and METEOR scores of these three models, it is easy to find that all of them have the same overall trend. BLEU-1 and

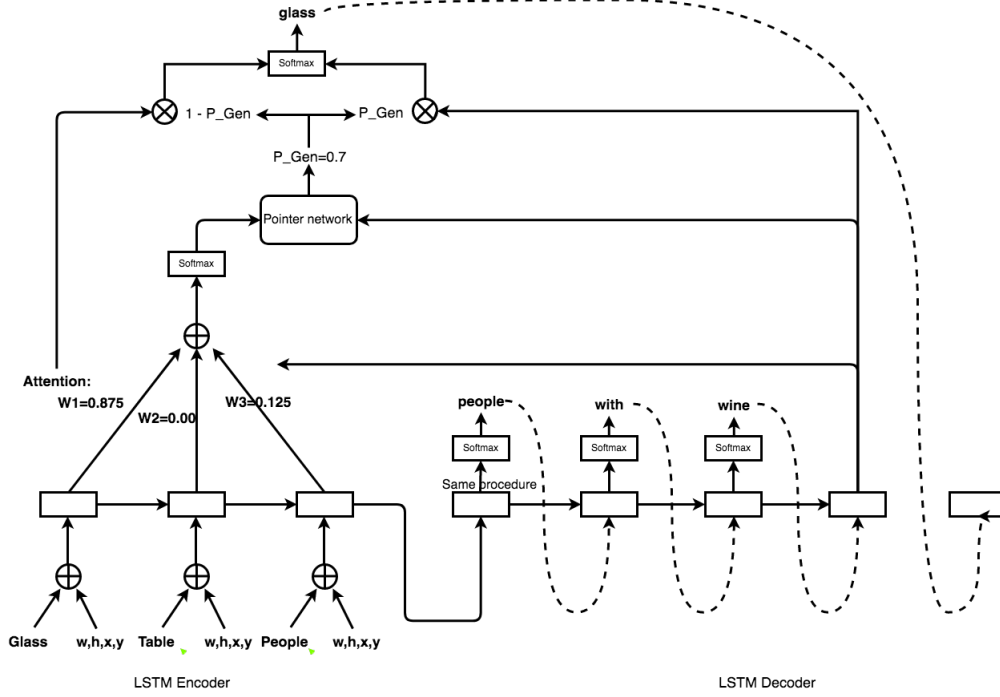


Figure 2: Structure of pointer network approach

Model	BLEU 1	BLEU 2	BLEU 3	BLEU 4	METEOR	ROUGE-L	CIDEr
Baseline	0.690	0.507	0.361	0.258	0.234	0.503	0.857
Pointer-Generator	0.688	0.506	0.358	0.252	0.232	0.501	0.846
Google Transformer	0.700	0.520	0.373	0.265	0.235	0.509	0.875

Table 1: Results on MS-COCO

ROUGE-L scores have larger variances and most of the samples fall in the range of 0.5 to 0.9 for BLEU-1, and 0.3 to 0.7 for ROUGE-L. CIDEr and METEOR scores are more concurrent and the majority of samples range from 0 to 2 for CIDEr, and 0.1 to 0.4 for METEOR.

4.2 Qualitative Results

Although the quantitative results are highly similar, the difference between the quality of the visual descriptions generated by different models are more evident. In this section, we will show some instances to explain the improvements.

In Table 2, we showed some improvements made by the pointer network model. By observation, we can find that the pointer network can deal with the number of objects better, which may contributed from the fact that it can directly retrieve object counts.

And in Table 3, we showed the improvements made by the Google Transformer based model. As we can see, this model learns a better bias that,

in standard cases, has higher generalizability. For example, in the last image shown in Table 3, the model applies a bias to caption the image as a pizza with many toppings, when in fact the pizza has no toppings. However, it is rare to have a pizza without toppings, which indicates that the bias is desirable. Given that there is no visual input to the model (i.e. the lack of an object categorized as topping), it can only guess that this unorthodox pizza without any toppings would also have a lot of toppings, which is obviously wrong in the given image. This is likely due to the fact that the network itself is deeper, and more information is stored in the model.

4.3 Future Work

Incorporation of visual feature. To further improve the quality of generated visual descriptions, it is reasonable to use visual features as additional input on top of object labels and locations. In image captioning tasks, OBJ2TEXT-YOLO-CNN/RNN incorporates visual features

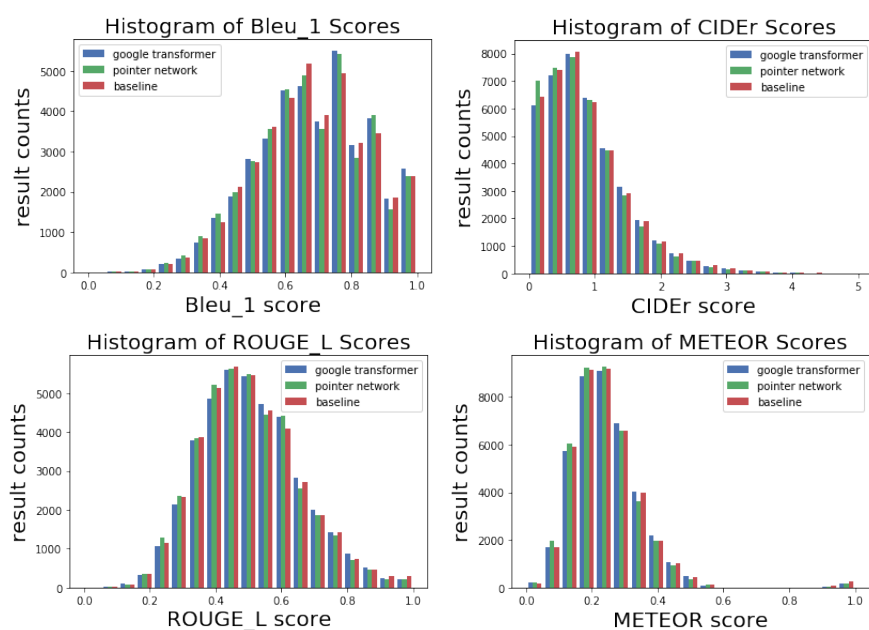


Figure 3: Histograms for four different scores of the three model




Example Images	
	<p>bsl-predict: a large passenger jet sitting on top of an airport tarmac</p> <p>ptr-predict: a group of airplanes are parked on the runway</p> <p>reference: several planes lined up on a runway during a cloudy day</p>
	<p>bsl-predict: a man holding a tennis racquet on a tennis court</p> <p>ptr-predict: two women playing tennis on a tennis court</p> <p>reference: two girls play a game of tennis on a sunny day</p>
	<p>bsl-predict: a boat with a large number of <i>unk</i> on it</p> <p>ptr-predict: a boat in the water with people in the background</p> <p>reference: a white yacht going down a river</p>

Table 2: Comparison of the results from pointer network model and baseline model


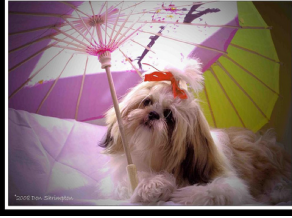

Example Images	
	bsl-predict: a bus is parked in the grass near a building gtf-predict: a large bus parked on a city street reference: a white bus that has a 41 crosstown sign on the front
	bsl-predict: a dog with a red umbrella sitting on the ground gtf-predict: a dog is sitting under a umbrella in the sun reference: a dog with a umbrella that is pointing to the sky
	bsl-predict: a pizza sitting on top of a white plate gtf-predict: a pizza with a lot of toppings on it reference: a large cheese pizza on a white plate

Table 3: Comparison of the results from google transformer model and baseline model

from VGG-16 and significantly improves the performance as compared to OBJ2TEXT-YOLO. However, the connection with VGG-16 does not explicitly utilize the object layout feature we already have. Instead, we propose a model that takes into account visual features from the YOLO object detector (?) and make it a part of the RNN language model (LSTM in our case) input, in addition to the object’s category and location. The structure of the model is shown in Figure 4. More specifically, the encoder at each time-step t takes as input $\langle o_t, l_t, v_t \rangle$, a tuple of the object’s category, object location and object visual features. o_t is category encoded in one-hot embedding of size V , $l_t = [B_t^x, B_t^y, B_t^w, B_t^h]$ is the object location configuration, and v_t is a vector of size 1024, which encodes the visual features of the cell where point (B_t^x, B_t^y) lands on YOLO. v_t is extracted from the YOLO architecture, but with the last two fully-connected layers dropped, and we only take the value in the corresponding cell as the visual features. o_t , l_t and v_t are mapped to the same size and added to form the input x_t to LSTM-based encoder as follows:

$$x_t = W_o o_t + W_l l_t + W_v v_t + b_l, x_t \in R^k$$

where $W_o \in R^{k \times v}$ is used to map o_t to R^k , $W_l \in R^{k \times 4}$ is used to map l_t to R^k , $W_v \in$

$R^{k \times 1024}$ is used to map v_t to R^k , and $b_l \in R^k$ is a bias variable. x_t is fed into LSTM with initial hidden state $h_0 = 0$.

Although we believe that our architecture as designed is promising to produce plausible results, the training of YOLO detector on the MS-COCO dataset led to out memory error (CPU) on Standard_NC6 Virtual Machine using Tesla K80 and 56GB CPU. After we switched to Standard_NC24 where the CPU memory is quadrupled to 224GB, the training still suffers from problems of segmentation faults and thus failed to converge. Our suspect is that the code is not optimized and thus takes up too much memory space when trained on a huge dataset like MS-COCO. The future research can be focused on the improvement of YOLO implementation, and experiments on the proposed model.

5 Conclusion

Based on Google Transformer and Pointer Network, we proposed two variants of OBJ2TEXT to generate better visually descriptive languages. On the MS-COCO dataset, we achieved higher scores in all evaluation metrics using Google Transformer based model, and equivalent results using Pointer Network based model.

The improvement of results and a better learned

