SLML final project: Image captioning

Abstract—In this report, we mainly learned the from the work in [3] and [2]. The two ariticle detailed the end-to-end model for image captioning. Particularly, the latter implements attention mechanism, and visualizes attention transition within a sentence. Our work are trials on reproduction of the previous model, building an recurrent neural network with captions in Flickr30k dataset and the ready-made image features. We slightly change the model structure, making use of GRU memory block and stacking up more layers. Then we use the feature extracted from convolution layer, split them into different spacial location and form into an attention context. Conducting experiments on the senctence generation along with along with attention visualization. We have revealed some success in our work. But on the whole, the performance is not satifactisfying. Therefore we pose some discussions on our work for future improvement.

Index Terms—Image captioning, End-to-end model, attention.

1 Basic model

[3] proposed an end-to-end model generating discription from images and achieve state-of-art performance. The end-to-end model use CNN as the encoder, extracting information from the image, and use RNN as the decoder, generating captions from the network. Our work are basically trials to reproduce [3], with the help of open codes from https://github.com/Hvass-Labs/TensorFlow-Tutorials.

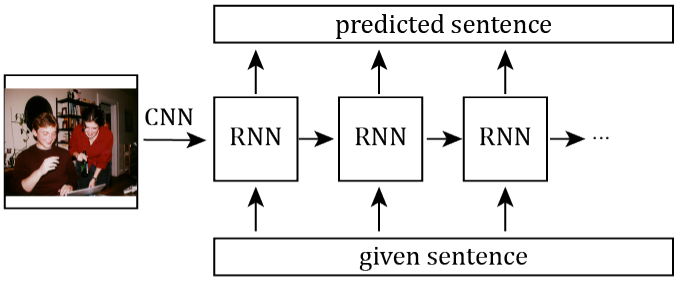

Fig. 1: *General framework of the end-to-end image captioning model. The model use a CNN as the encoder and a RNN series as the decoder. This figure shows the general framework of model training.*

1.1 Image feature and word embedding

Image features are usually extracted by a deep convolution neural network, such as VGG-net, Google-net, Resnet. In this report we simply adopted the ready-made resnet features, which are 2048-dimensional vectors. The extracted features cannot be use imediately. One more dense layer is required, transfering the feature to a vector, fed as the initial state of RNN.

The pre-processing of words is tokenizing and embedding. Tokenizing number the words in the vocabulary with positive integers. It should be noted that two special tokens are also included ? the start token numbered as 2 and the end token numbered as 3. Punctuation marks are excluded from the vocabulary. The tokenizing procedure is implemented by `Tokenizer` from module `tensorflow.keras`.

The following word embedding maps the word tokens to a continuous vector space. One-hot word embeddingc scheme has some applications in sentence generation, but is not suitable here, for the word based caption task which needs a huge vocabulary. A surprising fact is that with a well-designed continouous word embedding scheme, algebraic computations on embedding vectors could reflect the relations between words. Another method of continuous word embedding is built in the model. That is to say, it is a dense layer trained together with the whole task. This built-in type word embedding lacks the surprising algebraic properties, but is more adaptive to the certain model.

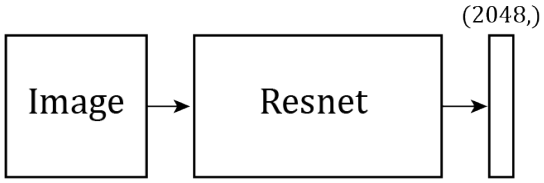

Fig. 2: *Image feature extraction. We use a ready-made resnet feature with a size of 2048.*

1.2 Detailed structure

The detailed model structure of training procedure is illustrated in Figure 4. The transfered image feature are fed into RNN as the initial state.

At each time step, an embedded word is also fed into RNN as the input value. The dense layer on the top transfers the RNN outputs to vector with the same size with the vocabulary. The transfered outputs are called logits, and later on ther are activated by softmax function, turned into a discrete probability distribution, indicating how likely to generate each word.

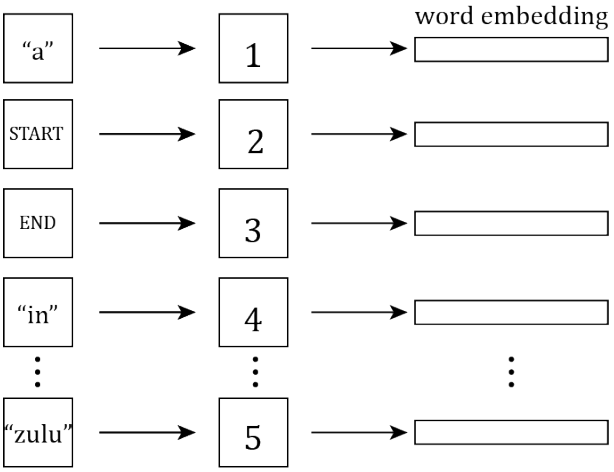

Fig. 3: *Word tokenizing and embedding. Words, including a start and an end token, are first tokenized as positive intergers. Then they are embedded into a continuous vector space.*

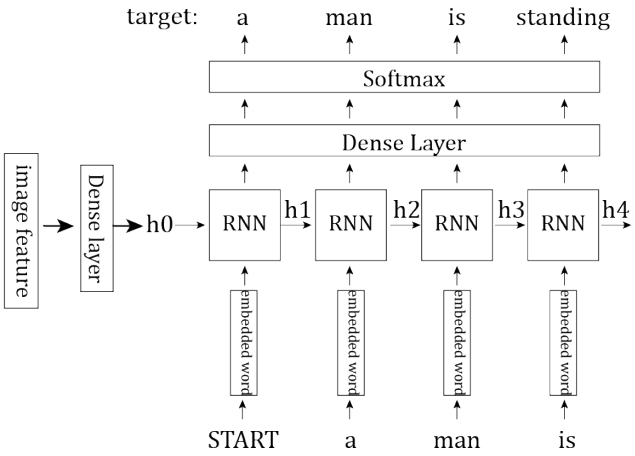

Fig. 4: *Detailed model structure. The figure shows the input and output of a training procedure.*

1.3 Settings

1.3.0.1 RNN blocks: Two kinds of RNN blocks are usually used, LSTM and GRU. They are designed for the same purpose of dealing with vanishing gradients in long sequence modeling. They both contain a mechanism called memory cell, in order to maintain the information during a long-distance transfering.

The model proposed by [3] uses LSTM memory block. LSTM's memory cell is controlled by three gates, namely *input*, *output* and *forget*. The three gates decided whether the memory cell should read the input value, whether to output the memory cell and whether to dump away the memory cell value. The mathematical definition of LSTM is

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1} + b_i) \\
f_t &= \sigma(W_{fx}x_t + W_{fm}m_{t-1} + b_f) \\
o_t &= \sigma(W_{0x}x_t + W_{om}m_{t-1} + b_o) \\
c_t &= f_t * c_{t-1} + i_t * h(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \\
m_t &= o_t * c_t \\
p_{t+1} &= \text{Softmax}(m_t)
\end{aligned}
$$

where t denote the time step, $i,\ f,\ ,o$ respectively denotes input, forget and output gate. c is the memory cell and p denotes the probability. σ, h are activation functions, generally chose to be sigmoid and hyperbolic tangent. $*$ denotes the elementwise product. All the W's and b's here are trainable parameters.

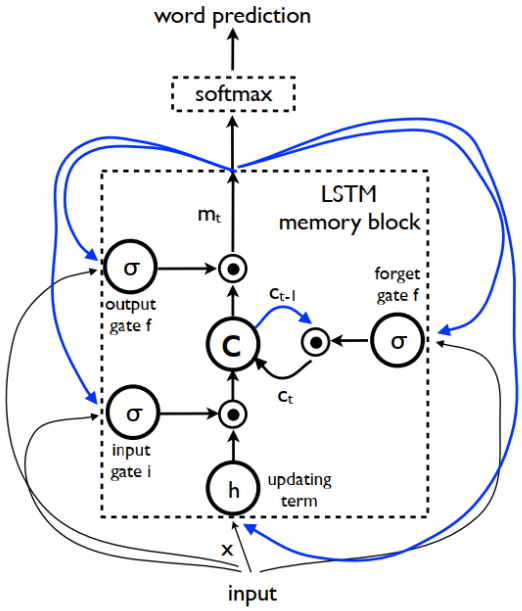

Fig. 5: *LSTM memory block. Retrieved from [3], page 3.*

The alternative GRU is deemed as a variant of LSTM. Instead of using three gates, GRU mix input gate and forget gate together as a update gate. The mathematical definition of GRU is

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * h(W_h x_t + U_h(r_t * h_{t-1}) + b_h)
\end{aligned}
$$

where z_t, r_t respectively denote the update gate and the reset gate.

While there has been so much tricky design of LSTM and its varaints' memory cell, [1] shows that their performance are all about the same.

1.3.0.2 Stack up RNNs: We found that appropriately adding more layers of RNN yields better decrease in loss function. Running 50 epochs, three layers of GRU can produce 0.1 more decreasing.

1.3.0.3 Hyperparameters: We choose RMSprop as the optimizer, the learning rate set as 0.001. We a RNN state size of 512 and an embedding size of 128.

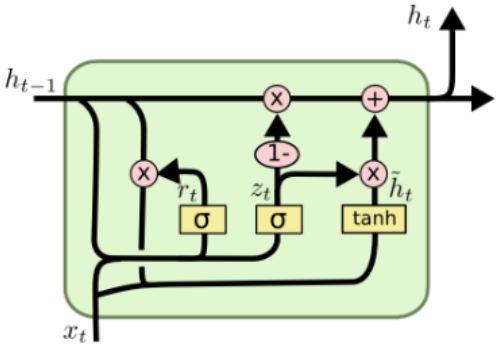

Fig. 6: *GRU memory block. GRU merges the input and output gate in LSTM together forming an update gate (z_t). Another gate is reset gate (r_t). Retrieved from colah.github.io/posts/2015-08-Understanding-LSTMs*

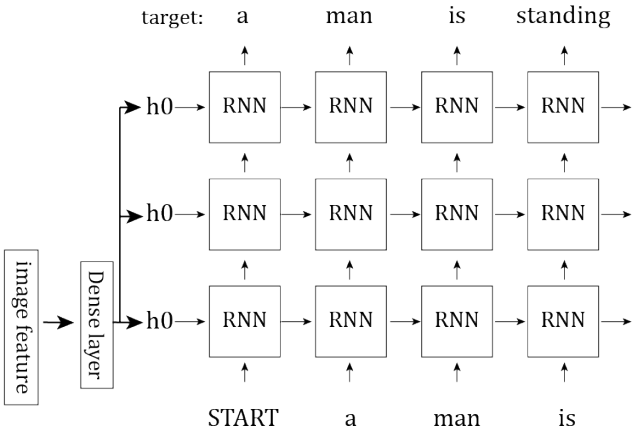

Fig. 7: *Stack three layers of RNN.*

1.4 Sentence generation

The inference framework has one difference with training. Instead of feeding the whole true caption as the sequence input, the inference model only has transefered image feature and a start token as input, then generating the first predicted word from the first RNN block's output. At each time step iteratively, the current RNN block takes the last predicted word as the block's input and repeat the procedure.

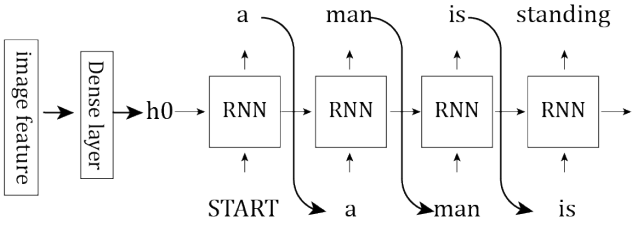

Fig. 8: *Model framework for inference. Different from training, inference use the word generate in the last time step as the input.*

There are basically three ways for word prediction from the discete probability distribution ? max probability, sampling and beam search. With several trials, we found that beam search performs the best. Max probability method, which simply takes the most likely word, is less efficient but very simple to implemented. Comments says that max probability method has the risk of stucking in a loop, therefore usually choose sampling method instead. However we found that with a large temperature the sampling method often generate worse discriptions, and with a small temperature, it behaves with no difference from max probability method.

1.5 Performance

9 shows and example of generated caption. This is a relatively successful example, where generatated discriptions do have obvious relation with the image, and the discriptions themselves are basically fluent sentences.

Ture caption:
a shirtless skateboarder is in midair, flying over a small, colorful ramp.
Generated caption:
1. a boy jumps into a water puddle of water
2. a boy does a skateboarding trick down a short flight kick outdoor

Fig. 9: *An example of generated caption. 1. is generated by beam search and 2. is generated by max probability. Both generation has some obvious relation with the gist of the picture, and they are basically fluent sentences. But their failures are also explicit. The first discription regard the ramp as a water puddle, although the two objects share same color. The second discription writes "down a short flight kick", which makes no sense, even that the boy's action has some similarities with a flight kick.*

However, failed examples, as shown in Table 1, are also easy to find. A big issue is that, the model seems to be more capable of capture sentence structure, instead of the what are exactly contained in the images. The outcome is for most of the time the model generates meaningful sentences, while only a few times do the discriptions have strong relations to the image.

2 Image Captioning with Attention

2.1 Introduction to attention model

In previous work, we use features of the complete image. In this section, we add the attention mechanism.

Predicted caption	**True caption**
a man in a white shirt and black vest is playing a guitar and singing with a microphone.	a woman in a long black dress playing the violin with a black man in a striped shirt , while the other band member with guitar plays along.
a man in a red shirt and a man in a red shirt are dancing in a parking lot with a crowd of onlookers behind them.	two people are wearing a chinese dragon costume in front of an ornate chinese gazebo with red pillars and the gazebo has two large red lanterns hanging from the roof.
a man in a white shirt is standing on a ladder cleaning a window.	a man in a white shirt kneels in the doorway of a building while two men behind him look on.

TABLE 1: *Some failed generations.*

In cognitive, selective attention demonstrates how we pay our attention to concrete objects. It helps us tune out irrelevant information and concentrate on what is real important.

The biggest difference is that original model treating the whole image equally while attention model pays attention to concrete areas. We make the first word prediction with our first attention area. We then shift and explore new attention area to predict next words with hidden state.

Basically, we change the image features in original model to a new attention mechanism model.

From

$$h_t = f(x, h_{t-1})$$

to

$$h_t = f(attention(x, h_{t-1}), h_{t-1})$$

Use a simple picture to show the change.

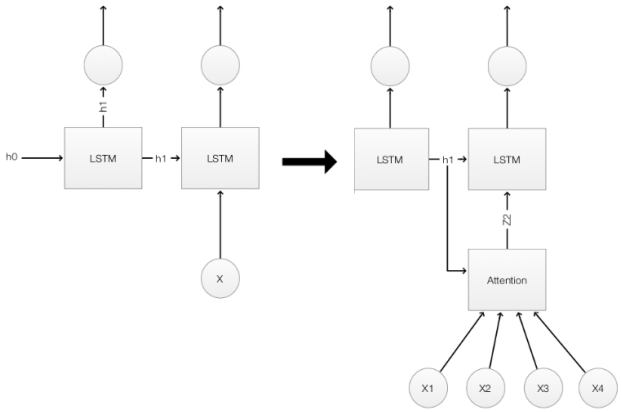

Fig. 10: *The change after adding attention mechanism. Instead of using the embedded word as the single input, LSTM with attention compute an "attention" combine the weighted sum of features from different location along with the input word. Retrieved from https://jhui.github.io/2017/03/15/Soft-and-hard-attention/*

We make a reproduction about attention model with soft attention soft mechanism of [2].

2.2 Details in attention mechanism

The attention models has 2 inputs:the hidden state(h_t) and image features in each localized ares(x) and 1 output:context vector(z).

We are supposed to use a convolutional neural network to extract features to feed attention model. However, we directly use the existing data sets to work. The attention feature of each image produces 2048 vectors corresponding to a part of the image.

$$x =< x_1, x_2, \cdots, x_{2048} >$$

Briefly, z_t shows the corresponding parts of image which is input at time t. There are different mechanisms Φ computing z_t from the feature x(2048 vectors). For each vector which maps a different part of image, the mechanism generates weight $weighted_i$ to represents relative importance to part i coupled with x_i(soft attention) or the probability that part i is the correct place to focus for predicting next words(hard attention).

[2] presented the formula

$$\begin{aligned} \alpha_{ti} &= f(x_i, h_{t-1})(attention\ model), \\ weighted_{ti} &= softmax(\alpha_{ti}), \\ z_t &= \Phi(x, weighted)(attention\ mechanism). \end{aligned}$$

The LSTM model has sight changes. We follow [4]:

$$\begin{aligned} i_t &= \sigma(W_i Ey_{t-1} + U_i h_{t-1} + Z_i z_t + b_i), \\ f_t &= \sigma(W_f Ey_{t-1} + U_f h_{t-1} + Z_f z_t + b_f), \\ c_t &= f_t c_{t-1} + i_t tanh(W_f Ey_{t-1} + U_f h_{t-1} + Z_f z_t + b_c), \\ o_t &= \sigma(W_o Ey_{t-1} + U_o h_{t-1} + Z_o z_t + b_o), \\ h_t &= o_t tanh(c_t). \end{aligned}$$

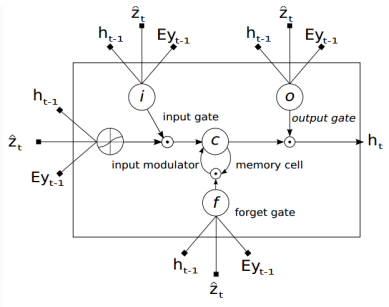

Fig. 11: *LSTM with attention. We can see from the figure that, the main difference from the math formula of a plain LSTM is adding one more term, Ey_{t-1}, into the inputs. The added term is the attention, or say weighted sum of features from different locations. Retrieved from https://jhui.github.io/2017/03/15/Soft-and-hard-attention/*

We use a table and a figure to show mentioned parameters.

2.3 Soft and hard attention

Although we only use soft attention mechanism. We will introduce two different mechanisms.

Notation	Definition	Notation	Definition
y	encoded words	W	weight matricie
i_t	input cell	U	weight matricie
f_t	forget cell	Z	weight matricie
c_t	memory cell	b	biases
o_t	output cell	h_t	hidden state

In soft attention, high attention area keeps the original value while low attention areas get closer to 0. $z = \sum weighted_i x_i$

In hard attention, weighted vector is used as sample rate to pick one x_i as the input to the LSTM. A deterministic method is replaced by a stochastic sampling model. $z \sim x_i, weighted_i$

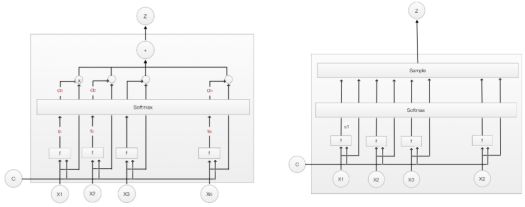

Fig. 12: *Soft attention vs. hard attention. Soft attention uses the weighted sum of all features, while hard attention samples from them. Retrieved from https://jhui.github.io/2017/03/15/Soft-and-hard-attention/*

2.4 Visualization of image attention

We use a visual python file to show our attention work for one image.

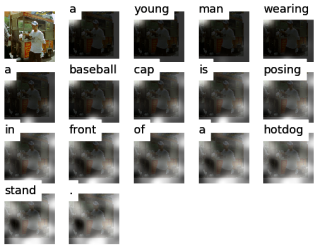

Fig. 13: *Visualization of the attention.*

3 Results

We may not control epoch and learning rate perfectly, so we do not get a good result.

Basic model		Attention model	
CIDEr	0.192	CIDEr	0.288
Bleu 1	0.382	Bleu 1	0.543
Bleu 2	0.232	Bleu 2	0.325
Bleu 3	0.171	Bleu 3	0.247
Bleu 4	0.141	Bleu 4	0.204

TABLE 2: *Coco-caption scores. Temporal scores are low and futher improvements are required.*

4 Discussion

Generally, we have completed the model, and observed substantially decreases in loss funtion during training. However the captioning performance is still far from perfect.

In our basic model, the loss get stucked at the level about 0.6. A large proportion of the generated sentences only have week correlation with the image. We haven't detect the problem that prevent the model from further training. We guess that it has something to do with the model structure. [3] claims that only input the image feature at the beginning of the recurrent network perform better than input it to every nodes. However we want to rethink on this idea due the fact that image feature does not play an important role in our model.

In attention model, results don't focus on particular objects which correspond to words. We think the reason may be a image is divided to two many parts, consequently, our model works poorly.

In future work, we think we should use CNN to recognize objects such as scene, action and so on in images to extract important information. Combine information to image information vector to be put in our model instead of using feature which is extracted from the whole image.

References

[1] e. a. Greff, Klaus. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, pages 2222–2232.

[2] R. K. K. C. A. C. R. S. R. Z. Y. B. Kelvin Xu, Jimmy Ba. Show, attend and tell: Neural image caption generation with visual attention. 2016.

[3] S. B. Oriol Vinyals, Alexander Toshev. Show and tell: A neural image caption generator. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164.

[4] I. S. Zaremba, Wojciech and O. Vinyals. Recurrent neural network regularization. 2016.