Seq2Seq with Attention and Beam Search

1. Introduction

This post is the first in a series about im2latex:Seq2Seq for LaTeX generation: its goal is to cover the concepts of Sequence-to-Sequence models with Attention and Beam search.
You may have heard from some recent breakthroughs in Neural Machine Translation that led to (almost) human-level performance systems (used in real-life by Google Translation, see for instance this paper enabling zero-shot translation). These new architectures rely on a common paradigm called encoder-decoder (or sequence to sequence), whose goal is to produce an entire sequence of tokens.

2. Sequence to Sequence basics

> As an example, let's translate how are you in French comment vas tu.

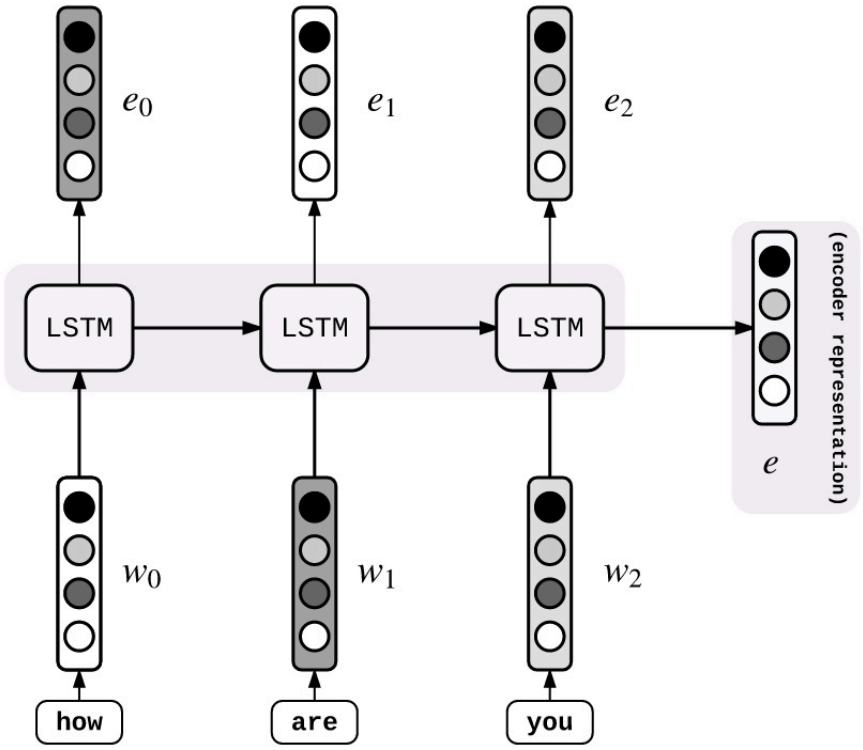
2.1 Vanilla Seq2Seq

The Seq2Seq framework relies on the encoder-decoder paradigm. The encoder encodes the input sequence, while the decoder produces the target sequence

Seq2Seq依赖于encoder-decoder模型。

> **Encoder**

Our input sequence is `how are you`. Each word from the input sequence is associated to a vector $w \in \mathbb{R}^d$ (via a lookup table). In our case, we have 3 words, thus our input will be transformed into $[w_0, w_1, w_2] \in \mathbb{R}^{d \times 3}$. Then, we simply run an LSTM over this sequence of vectors and store the last hidden state outputed by the LSTM: this will be our encoder representation e. Let's write the hidden states $[e_0, e_1, e_2]$ (and thus $e = e_2$)

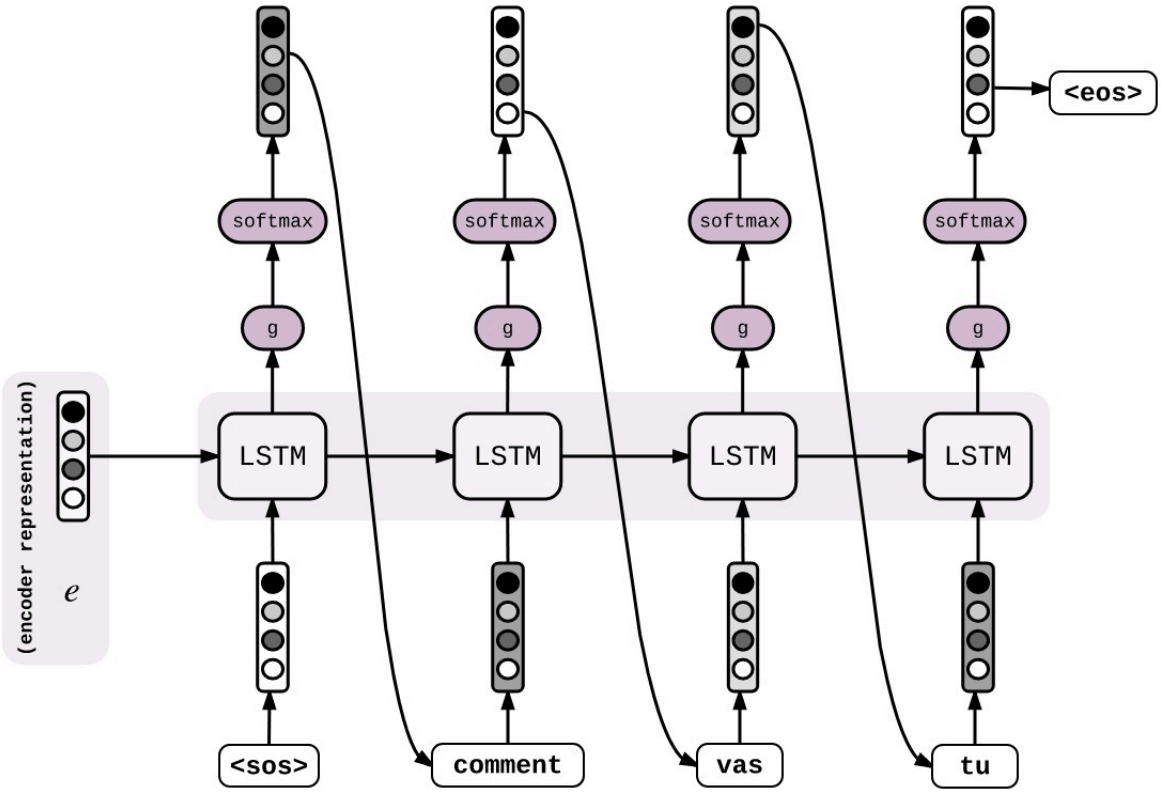


Vanilla Encoder

简而言之：输入x_t为词向量,$[w_0, w_1, w_2]$为矩阵参数，LSTM将最终的$e = h_{end}$作为最终的encoder representation

> **Decoder**

Now that we have a vector e that captures the meaning of the input sequence, we'll use it to generate the target sequence word by word. Feed to another LSTM cell: e as hidden state and a special start of sentence vector $wsos$ as input. The LSTM computes the next hidden state $h_0 \in \mathbb{R}^h$. Then, we apply some function $g : \mathbb{R}^h \mapsto \mathbb{R}^V$ so that $s_0 := g(h_0) \in \mathbb{R}^V$ is a vector of the same size as the vocabulary.



Vanilla Decoder

简言之：Encoder生成的向量e和$< sos >$作为Decoder的输入，LSTM输出$g = W_g \cdot h_t$，再经过softmax后输出整个vocabulary中每个词对应的概率，选择最大概率对应的单词的embedding作为下一个时刻LSTM的input。而最大单词的索引i_t即为该时刻翻译成的单词。

$$
\begin{aligned}
h_t &= \mathrm{LSTM}(h_{t-1}, w_{i_t}) \\
s_t &= g(h_t) \\
p_t &= \mathrm{softmax}(s_t) \\
i_t &= \mathrm{argmax}(p_t)
\end{aligned}
$$

以上过程直至出现特定的`end of sentence token`($< EOS >$)停止。综上所述

> Intuitively, the hidden vector represents the “amount of meaning” that has not been decoded yet. Decoder的隐层vector代表还未decoder的信息的剩余价值。

The above method aims at modelling the distribution of the next word conditionned on the beginning of the sentence.

$$\mathbb{P}\left[y_{t+1} | y_1, \ldots, y_t, x_0, \ldots, x_n\right]$$

by writing

$$\mathbb{P}\left[y_{t+1} | y_t, h_t, e\right]$$

也就是说Decoder模型就是根据句子前文的单词对下一个单词进行数据分布的建模。

2.2 Seq2Seq with Attention

Attention is a mechanism that forces the model to learn to focus (=to attend) on specific parts of the input sequence when decoding, instead of relying only on the hidden vector of the decoder's LSTM.One way of performing attention is explained by Bahdanau et al.. We slightly modify the reccurrence formula that we defined above by adding a new vector c_t to the input of the LSTM.
推荐Attention的参考资料：Attention and Augmented Recurrent Neural Networks
根据Attention机制，给出如下公式，其中$w_{i_{t-1}}$是前一个word，h_t是前一个隐层状态

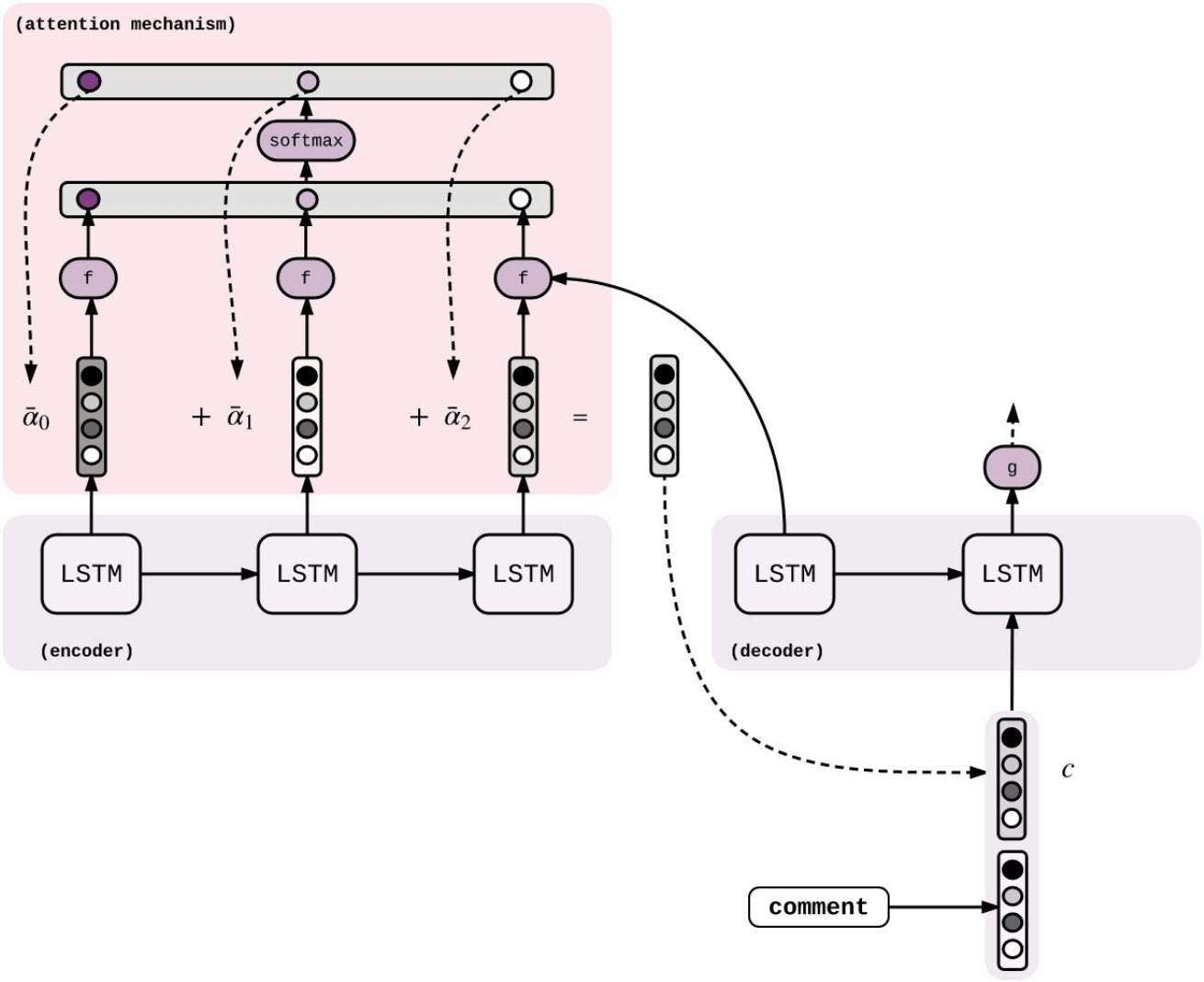
$$
\begin{aligned}
h_t &= \mathbf{LSTM}(h_{t-1}, [w_{i_{t-1}}, c_t]) \\
s_t &= g(h_t) \\
p_t &= \mathbf{softmax}(s_t) \\
i_t &= \mathbf{argmax}(p_t)
\end{aligned}
$$

上式中，形式和普通的Seq2Seq Decoder模型一致，区别在于输入增加了attention部分c_t，其计算方式如下：

$$
\begin{aligned}
\alpha_{t'} &= f(h_{t-1}, e_{t'}) \in \mathbb{R} \qquad \text{for all } t' \\
\bar{\alpha} &= \mathbf{softmax}(\alpha) \\
c_t &= \sum_{t'=0}^{n} \bar{\alpha}_{t'} e_{t'}
\end{aligned}
$$

The vector ct is the attention (or context) vector. We compute a new context vector at each decoding step. First, with a function $f(h_{t-1}, e_{t'}) \mapsto \alpha_{t'} \in \mathbb{R}$, compute a score for each hidden state $e_{t'}$ of the encoder. Then, normalize the sequence of $\alpha_{t'}$ using a softmax and compute c_t as the weighted average of the $e_{t'}$.

解释: 其中，$e_{t'}$是encoder的t'时刻的输出,h_{t-1}是decoder的前一个hidden state，c_t作为decoder的input之一。相比较without Attention version，Attention版本的每个时刻增加了一个变换的输入c_t，而在普通seq2seq中，不具有该输入（或者将该输入看成恒定值e，参照公式$\mathbb{P}\left[y_{t+1} | y_t, h_t, e\right]$）



Attention Mechanism

The choice of the function f varies, but is usually one of the following:

$$f(h_{t-1}, e_{t'}) = \begin{cases} h_{t-1}^T e_{t'} & \text{dot} \\ h_{t-1}^T W e_{t'} & \text{general} \\ v^T \tanh(W[h_{t-1}, e_{t'}]) & \text{concat} \end{cases}$$

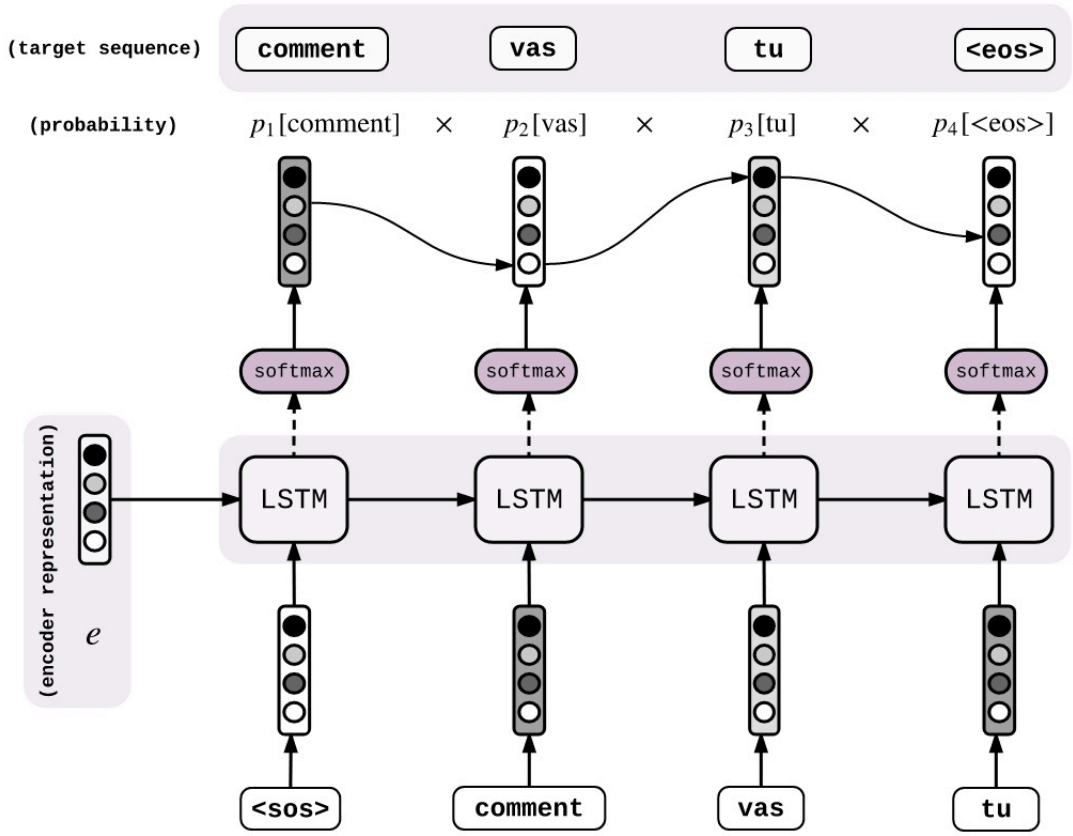
论文中f的选择是个DNN网络，类似于上述中的concat模式。

By putting the attention weights into a matrix (rows = input sequence, columns = output sequence), we would have access to the alignment between the words from the English and French sentences… (see page 6) There is still a lot of things to say about sequence to sequence models (for instance, it works better if the encoder processes the input sequence backwards…).

3. Training

> What happens if the first time step is not sure about wether it should generate comment or vas (most likely case at the beginning of the training)? Then it would mess up the entire sequence, and the model will hardly learn anything…

If we use the predicted token as input to the next step during training (as explained above), errors would accumulate and the model would rarely be exposed to the correct distribution of inputs, making training slow or impossible. To speedup things, one trick is to feed the actual output sequence (`<sos> comment vas tu`) into the decoder's LSTM and predict the next token at every position (`comment vas tu <eos>`).
简言之：训练的时候下一时刻的word输入是上一时刻的输出时，一旦前面单词预测错，后面的误差会越来越大，很难得到最终的正确分布。一个小trick就是把正确的单词作为下一时刻的word输入，这样能加速训练。



Training

The decoder outputs vectors of probability over the vocabulary $p_i \in \mathbb{R}^V$ for each time step. Then, for a given target sequence $y_1, \ldots, y_n$, we can compute its probability as the product of the probabilities of each token being produced at each relevant time step:

$$\mathbb{P}(y_1, \ldots, y_m) = \prod_{i=1}^{m} p_i[y_i]$$

上式就是单个sentence翻译正确的概率。得到交叉熵cross entropy损失，并最小化即可：

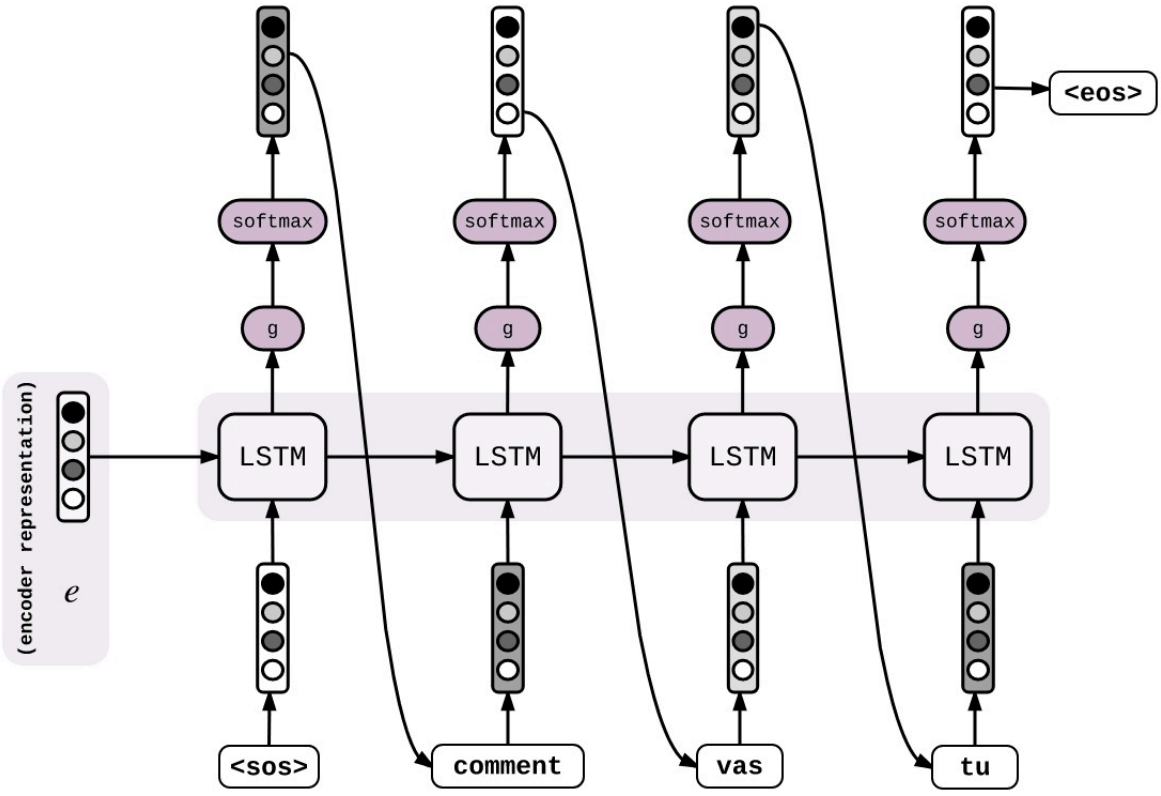
$$
\begin{aligned}
-\log \mathbb{P}\left(y_1, \ldots, y_m\right) &= -\log \prod_{i=1}^{m} p_i[y_i] \\
&= -\sum_{i=1}^{n} \log p_i[y_i]
\end{aligned}
$$

4. Decoding

训练过程使用了一个小trick加速训练。

> What about inference/testing time then? Is there an other way to decode a sentence?

greedy decoding
It is the most natural way and it consists in feeding to the next step the most likely word predicted at the previous step.



Greedy Decoder - feeds the best token to the next step

> But didn't we say that this behavior is likely to accumulate errors? 话说这种将上一个预测作为下一个预测的输入不是会积累误差么？所以这种方法本身就一般啊

Beam Search

Instead of only predicting the token with the best score, we keep track of k hypotheses (for example $k = 5$), we refer to k as the beam size). At each new time step, for these 5 hypotheses we have V new possible tokens. It makes a total of $5V$ new hypotheses. Then, only keep the 5 best ones, and so on… .
简单的说:

```
beam search操作属于贪心算法思想，不一定reach到全局最优解。只在test的时候需要。训练的时候知道正确答案，并不需要再进行这个搜索。
test的时候，假设词表大小为3，内容为a，b，c。beam size是2
decoder解码的时候：
1： 生成第1个词的时候，选择概率最大的2个词，假设为a,c,那么当前序列就是a,c
2：生成第2个词的时候，我们将当前序列a和c，分别与词表中的所有词进行组合，得到新的6个序列aa ab ac ca cb cc,然后从其中选择2个得分最高的，作为当前序列，假如为aa cb
3：后面会不断重复这个过程，直到遇到结束符为止。最终输出2个得分最高的序列。
```

> If we use beam search, a small error at the first step might be rectified at the next step, as we keep the gold hypthesis in the beam!

5. Conclusion

In this article we covered the seq2seq concepts. We showed that training is different than decoding. We covered two methods for decoding: greedy and beam search. While beam search generally achieves better results, it is not perfect and still suffers from exposure bias. During training, the model is never exposed to its errors! It also suffers from Loss-Evaluation Mismatch. The model is optimized w.r.t. token-level cross entropy, while we are interested about the reconstruction of the whole sentence…

Now, let's apply Seq2Seq for LaTeX generation from images!

Contributing

Please feel free to send me pull requests.

References

- https://guillaumegenthial.github.io/sequence-to-sequence.html
- https://www.zhihu.com/question/54356960