Paper Reading

卢宁 [1]

2020 年 11 月 8 日

目录

Table of Contents

什么是命名实体识别？

命名实体识别（NER）（也称为实体识别、实体分块和实体提取）是信息提取的一个子任务，旨在将文本中的命名实体定位并分类为预先定义的类别，如人员、组织、位置、时间表达式、数量、货币值、百分比等。

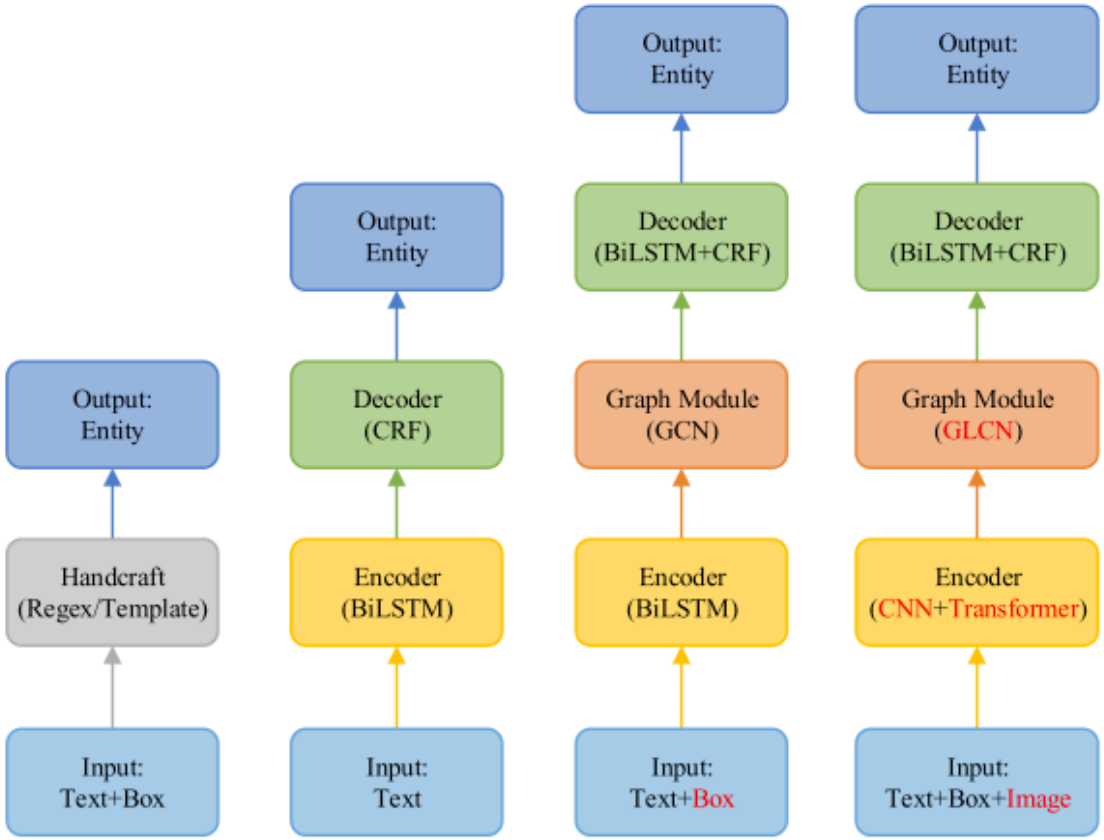
NER vs. Classification

1. NER 本质上是一个分类问题，只不过它一般针对序列。
2. localization + classification

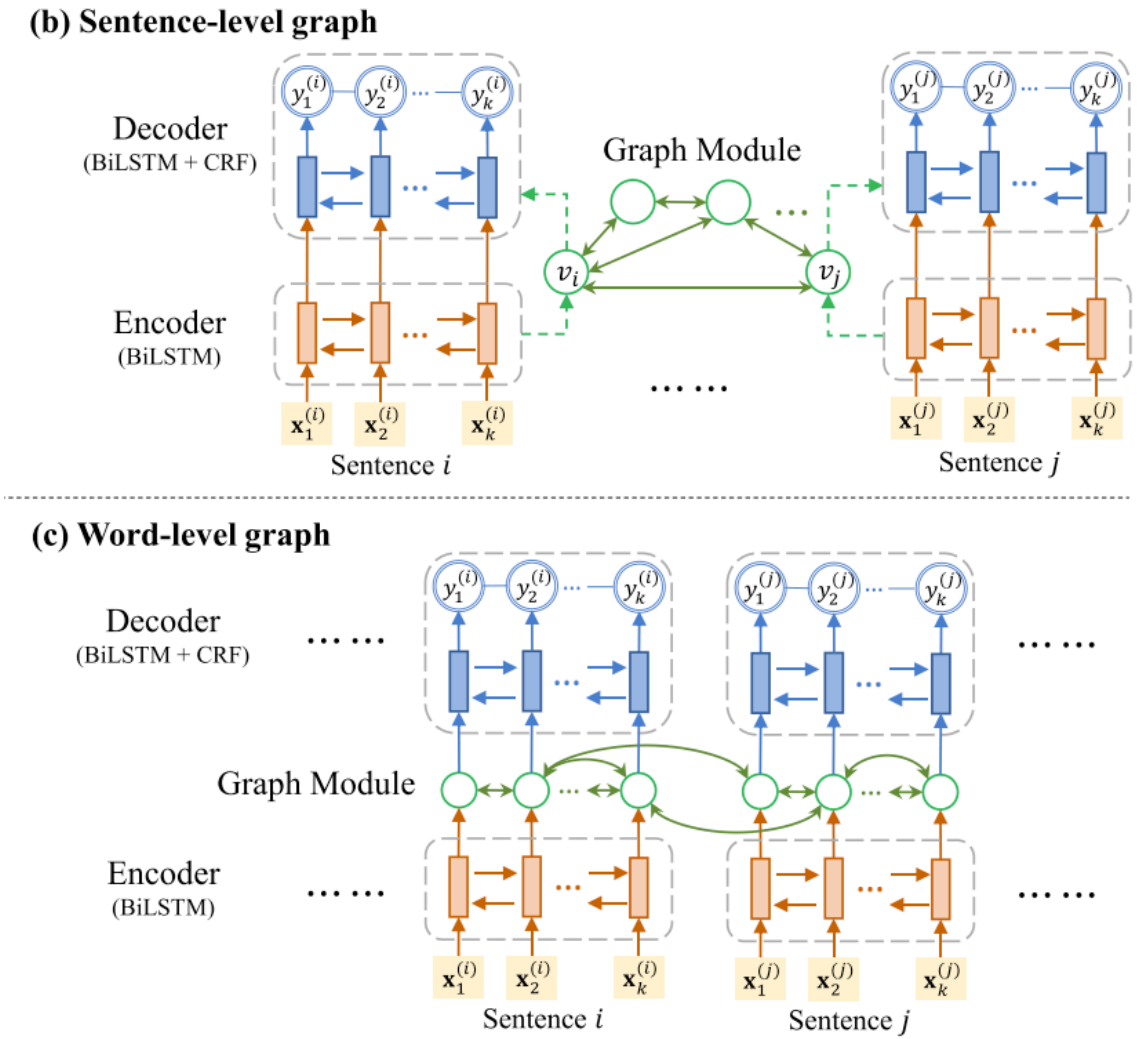
NER 的特点

1. 命名【命名实体】
   - 构词灵活
   - 类别模糊
2. 实体无穷
3. 歧义的消解
4. 边界的界定

NER 方法的演进


Output:
Entity
Handcraft
(Regex/Template)
Input:
Text+Box
Output:
Entity
Decoder
(CRF)
Encoder
(BiLSTM)
Input:
Text
Output:
Entity
Decoder
(BiLSTM+CRF)
Graph Module
(GCN)
Encoder
(BiLSTM)
Input:
Text+Box
Output:
Entity
Decoder
(BiLSTM+CRF)
Graph Module
(GLCN)
Encoder
(CNN+Transformer)
Input:
Text+Box+Image

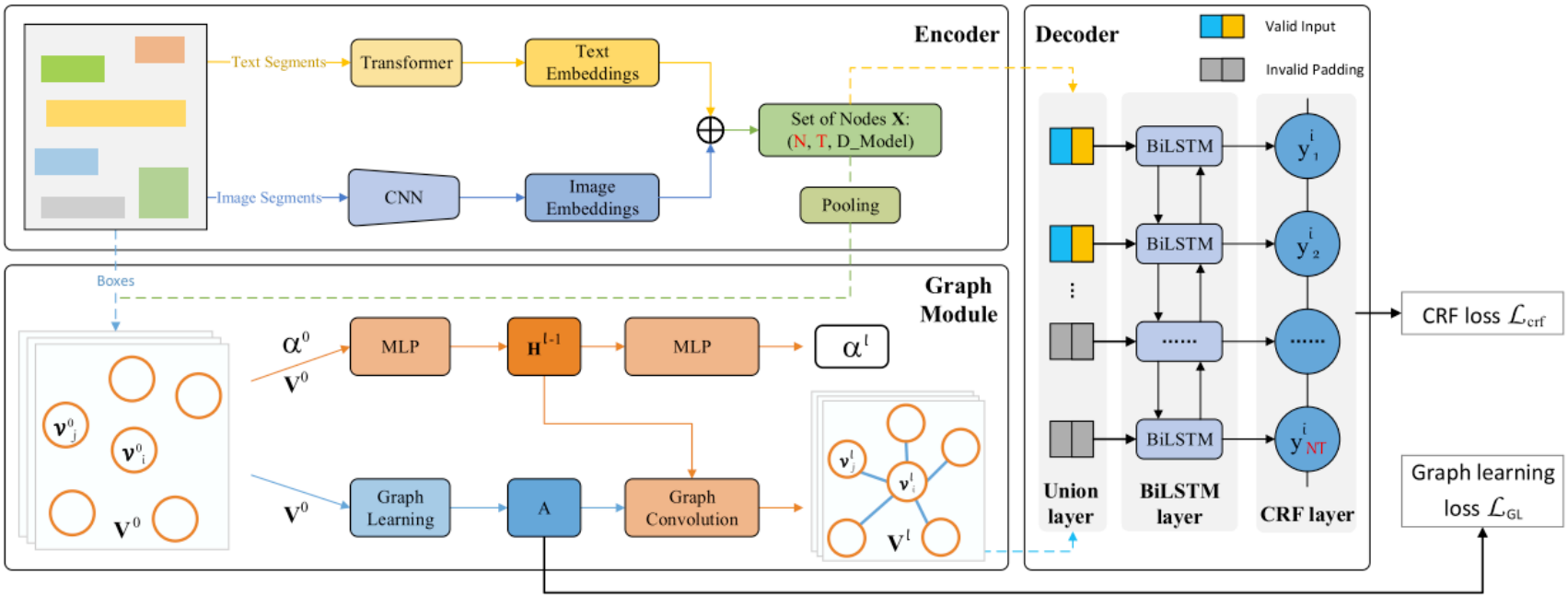
MIT 的工作


(b) Sentence-level graph
Decoder
(BiLSTM + CRF)
Encoder
(BiLSTM)
Graph Module
Sentence i
Sentence j
(c) Word-level graph
Decoder
(BiLSTM + CRF)
Graph Module
Encoder
(BiLSTM)
Sentence i
Sentence j

Alibaba 的工作


t6
t5
t2
r16
r12
r15
t1
r14
r13
t4
t3
t1 r12 t2
t1 r13 t3
t1 r14 t4
t1 r15 t5
t1 r16 t6
MLP
h12
h13
h14
h15
h16
t'1
hij
MLP
r'ij

我们方法


Encoder
Text Segments
Transformer
Text Embeddings
Image Segments
CNN
Image Embeddings
Set of Nodes X: (N, T, D_Model)
Pooling
Boxes
Graph Module
MLP
MLP
Graph Learning
A
Graph Convolution
Decoder
Valid Input
Invalid Padding
BiLSTM
BiLSTM
BiLSTM
Union layer
BiLSTM layer
CRF layer
CRF loss $\mathcal{L}_{crf}$
Graph learning loss $\mathcal{L}_{GL}$

Encoder

$$\mathbf{te}_{1:T}^{(i)} = \text{TransformerEncoder}\left(\mathbf{c}_{1:T}^{(i)}; \Theta_{\text{tenc}}\right) \tag{1}$$

$$\mathbf{TE} = \left[\mathbf{te}_{1:T}^{(1)}; \ldots; \mathbf{te}_{1:T}^{(N)}\right] \in \mathbb{R}^{N \times T \times d_{\text{model}}} \tag{2}$$

$$\mathbf{ie}^{(i)} = \text{CNN}\left(\mathbf{s_i}^{is}; \Theta_{\text{cnn}}\right) \tag{3}$$

$$\mathbf{IE} = \left[\mathbf{ie}^{(1)}; \ldots; \mathbf{ie}^{(N)}\right] \in \mathbb{R}^{N \times T \times d_{model}} \tag{4}$$

$$\mathbf{X} = \mathbf{TE} + \mathbf{IE} \tag{5}$$

$$\begin{cases} \mathrm{A}_{ij} = \mathrm{softmax}\left(\mathbf{e}_{ij}\right), \quad i = 1, \ldots, N, \quad j = 1, \ldots, N \\ \mathbf{e}_{\mathrm{ij}} = \mathrm{Leak\,Relu}\left(\mathbf{w}_i^T \left|v_i - v_j\right|\right) \end{cases} \tag{6}$$

$$\sum_{j=1}^{N} A_{ij} = 1, A_{ij} \geq 0 \tag{7}$$

$$\mathcal{L}_{\mathrm{GL}} = \frac{1}{N^2} \sum_{i,j=1}^{N} \exp\left(A_{ij} + \eta \left\|v_i - v_j\right\|_2^2\right) + \gamma \|\mathbf{A}\|_F^2 \tag{8}$$

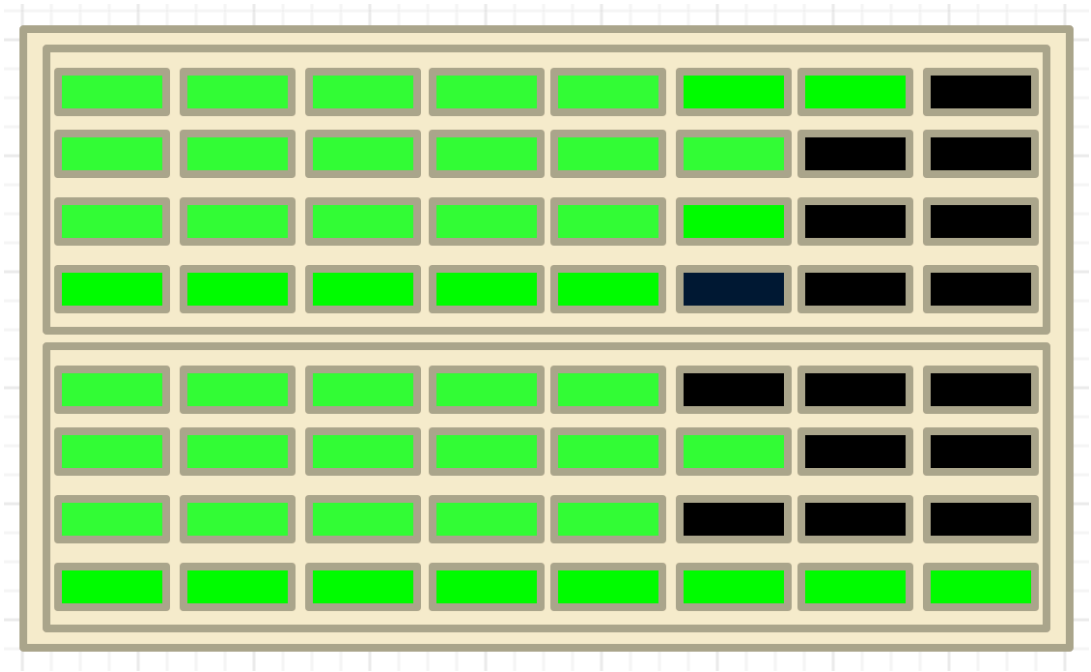
$$\mathbf{v}_i^{(l+1)} = \sigma\left(\mathbf{A}_i \mathbf{h}_i^l \mathbf{W}^l\right) \tag{9}$$

$$\alpha_{ij}^0 = \mathbf{W}_\alpha^0 \left[x_{ij}, y_{ij}, \frac{w_i}{h_i}, \frac{h_j}{h_i}, \frac{w_j}{h_i}, \frac{T_j}{T_i}\right]^T \tag{10}$$

$$h_{ij}^l = \sigma\left(\mathbf{W}_{v_i h}^l \mathbf{v}_i^l + \mathbf{W}_{v_j h}^l \mathbf{v}_j^l + \alpha_{ij}^l + \mathbf{b}^l\right) \tag{11}$$

$$\alpha_{ij}^{l+1} = \sigma\left(\mathbf{W}_\alpha^l \mathbf{h}_{ij}^l\right) \tag{12}$$

Decoder



Decoder

$$\mathbf{Z} = \mathrm{BiLSTM}\left(\hat{\mathbf{X}}; \mathbf{0}, \Theta_{\mathrm{lstm}}\right) \mathbf{W}_z \tag{13}$$

$$\begin{cases} \mathcal{L}_{\mathrm{crf}} = -\log(p(\mathbf{y}|\hat{\mathbf{X}})) = -s(\hat{\mathbf{X}}, \mathbf{y}) + Z \\ Z = \log\left(\sum_{\tilde{\mathbf{y}} \in \mathcal{Y}(\hat{\mathbf{x}})} e^{s(\hat{\hat{\mathbf{X}}}, \tilde{\mathbf{y}})}\right) = \underset{\tilde{\mathbf{y}} \in \mathcal{Y}(\hat{\mathbf{X}})}{\log \mathrm{dd}} s(\hat{\mathbf{X}}, \tilde{\mathbf{y}}) \end{cases} \tag{14}$$

$$\mathcal{L}_{\mathsf{total}} = \mathcal{L}_{\mathrm{crf}} + \lambda \mathcal{L}_{\mathrm{GL}} \tag{15}$$

实验细节

1. 8 V100 300 epoches , adam 0.0002
2. $\lambda = 0.01\ \eta = 1\ \gamma = 0.4$

消融实验

Model	Medical Invoice	Train Ticket
PICK (Full model)	87.0	98.6
w/o image segments	↓0.9	↓0.4
w/o graph learning	↓1.6	↓0.7

Table of Contents

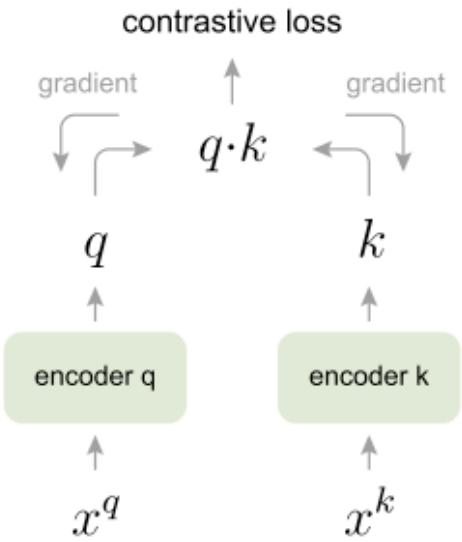
无监督和自监督表示学习

1. Instance-wise Contrastive Learning
2. Deep Unsupervised Clustering
3. Self-supervised Pretext Tasks.

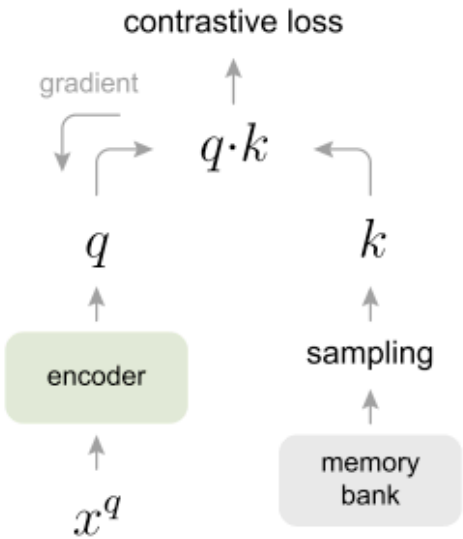
Table of Contents

Motivation

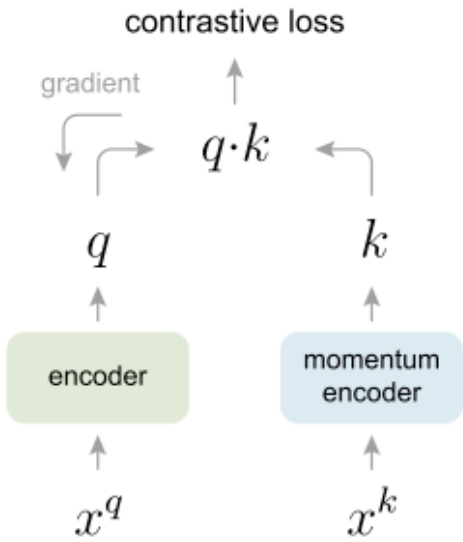
1. large
2. consistence



(a) end-to-end



(b) memory bank



(c) MoCo

Loss

$$\mathcal{L}_q = -\log \frac{\exp\left(q \cdot k_+ / \tau\right)}{\sum_{i=0}^{K} \exp\left(q \cdot k_i / \tau\right)} \tag{16}$$

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
   x_q = aug(x) # a randomly augmented version
   x_k = aug(x) # another randomly augmented version

   q = f_q.forward(x_q) # queries: NxC
   k = f_k.forward(x_k) # keys: NxC
   k = k.detach() # no gradient to keys

   # positive logits: Nx1
   l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

   # negative logits: NxK
   l_neg = mm(q.view(N,C), queue.view(C,K))

   # logits: Nx(1+K)
   logits = cat([l_pos, l_neg], dim=1)

   # contrastive loss, Eqn.(1)
   labels = zeros(N) # positives are the 0-th
   loss = CrossEntropyLoss(logits/t, labels)

   # SGD update: query network
   loss.backward()
   update(f_q.params)

   # momentum update: key network
   f_k.params = m*f_k.params+(1-m)*f_q.params

   # update dictionary
   enqueue(queue, k) # enqueue the current minibatch
   dequeue(queue) # dequeue the earliest minibatch
```

实验细节

1. ImageNet and Instagram-1B
2. $\tau = 0.07$ 224*224 crop from random resized image
3. random color jittering, hflip, grayscale conversion
4. SGD Weight Decay 0.0001 with momentum is 0.9
5. ImageNet 256 8 GPUs, lr is 0.03, 200 epoches, *0.1 120 160, 53 hours ResNet50
6. Instagram 1024 64 GPUs, lr 0.12, *0.9 62.5k (64M images), 1.25M (1.4 epoches), 6 days ResNet50

method	architecture	#params (M)	accuracy (%)
Exemplar [17]	R50w3×	211	46.0 [38]
RelativePosition [13]	R50w2×	94	51.4 [38]
Jigsaw [45]	R50w2×	94	44.6 [38]
Rotation [19]	Rv50w4×	86	55.4 [38]
Colorization [64]	R101*	28	39.6 [14]
DeepCluster [3]	VGG [53]	15	48.4 [4]
BigBiGAN [16]	R50	24	56.6
	Rv50w4×	86	61.3
methods based on contrastive learning follow:			
InstDisc [61]	R50	24	54.0
LocalAgg [66]	R50	24	58.8
CPC v1 [46]	R101*	28	48.7
CPC v2 [35]	$R170^{*}_{wider}$	303	65.9
CMC [56]	$R50_{L+ab}$	47	$64.1^{\dagger}$
	$R50w2\times_{L+ab}$	188	$68.4^{\dagger}$
AMDIM [2]	$AMDIM_{small}$	194	$63.5^{\dagger}$
	$AMDIM_{large}$	626	$68.1^{\dagger}$
MoCo	R50	24	60.6
	RX50	46	63.9
	R50w2×	94	65.4
	R50w4×	375	**68.6**

Table 1. **Comparison under the linear classification protocol on ImageNet**. The figure visualizes the table. All are reported as

pre-train	AP_{50}	AP	AP_{75}
random init.	64.4	37.9	38.6
super. IN-1M	81.4	54.0	59.1
MoCo IN-1M	81.1 (−0.3)	54.6 (**+0.6**)	59.9 (**+0.8**)
MoCo IG-1B	81.6 (+0.2)	55.5 (**+1.5**)	61.2 (**+2.1**)

(a) Faster R-CNN, R50-**dilated-C5**

pre-train	AP_{50}	AP	AP_{75}
random init.	60.2	33.8	33.1
super. IN-1M	81.3	53.5	58.8
MoCo IN-1M	81.5 (+0.2)	55.9 (**+2.4**)	62.6 (**+3.8**)
MoCo IG-1B	82.2 (**+0.9**)	57.2 (**+3.7**)	63.7 (**+4.9**)

(b) Faster R-CNN, R50-**C4**

Table 2. **Object detection fine-tuned on PASCAL VOC** `trainval07+12`. Evaluation is on `test2007`: AP_{50} (default VOC metric), AP (COCO-style), and AP_{75}, averaged over 5 trials. All are fine-tuned for 24k iterations (~23 epochs). In the brackets are the gaps to the ImageNet supervised pre-training counterpart. In green are the gaps of at least **+0.5** point.

Attention Mechanism 的原理

Language Model

$$p(y_i|y_1, ..., y_{i-1}, X) = g(y_{i-1}, s_i, c_i) \tag{17}$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{18}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{19}$$

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \tag{20}$$

$$e_{ij} = \alpha(s_{i-1}, h_j) \tag{21}$$

Attention Mechanism 的原理

Score Functions

$$score(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & \text{dot} \\ h_t^\top W_a \bar{h}_s & \text{general} \\ v_a^\top \tanh(W_a \left[h_t^\top; \bar{h}_s\right]) & \text{concat} \end{cases} \tag{22}$$

Attention Mechanism 的原理

Encoder

Table of Contents

Soft Attention vs. Hard Attention

1. Soft Attention 是参数化的（Parameterization），因此可导，可以被嵌入到模型中去，直接训练。梯度可以经过 Attention Mechanism 模块，反向传播到模型其他部分。
2. Hard Attention 是一个随机的过程。Hard Attention 不会选择整个 encoder 的输出做为其输入，Hard Attention 会依概率 Si 来采样输入端的隐状态一部分来进行计算，而不是整个 encoder 的隐状态。为了实现梯度的反向传播，需要采用蒙特卡洛采样的方法来估计模块的梯度。

Gloable Attention vs. Local Attention

1. 传统的 Attention model 一样。所有的 hidden state 都被用于计算 Context vector 的权重，即变长的对齐向量 at，其长度等于 encoder 端输入句子的长度
2. Global Attention 有一个明显的缺点就是，每一次，encoder 端的所有 hidden state 都要参与计算，这样做计算开销会比较大，特别是当 encoder 的句子偏长，比如，一段话或者一篇文章，效率偏低。因此，为了提高效率，Local Attention 应运而生。

Table of Contents

Why not RNN?

RNN（或者 LSTM，GRU 等）的计算限制为是顺序的，也就是说 RNN 相关算法只能从左向右依次计算或者从右向左依次计算，这种机制带来了两个问题：

- 时间片 t 的计算依赖 t-1 时刻的计算结果，这样限制了模型的并行能力；

Why not RNN?

RNN（或者 LSTM，GRU 等）的计算限制为是顺序的，也就是说 RNN 相关算法只能从左向右依次计算或者从右向左依次计算，这种机制带来了两个问题：

- 顺序计算的过程中信息会丢失，尽管 LSTM 等门机制的结构一定程度上缓解了长期依赖的问题，但是对于特别长期的依赖现象,LSTM 依旧无能为力。

经典回顾

Transformer

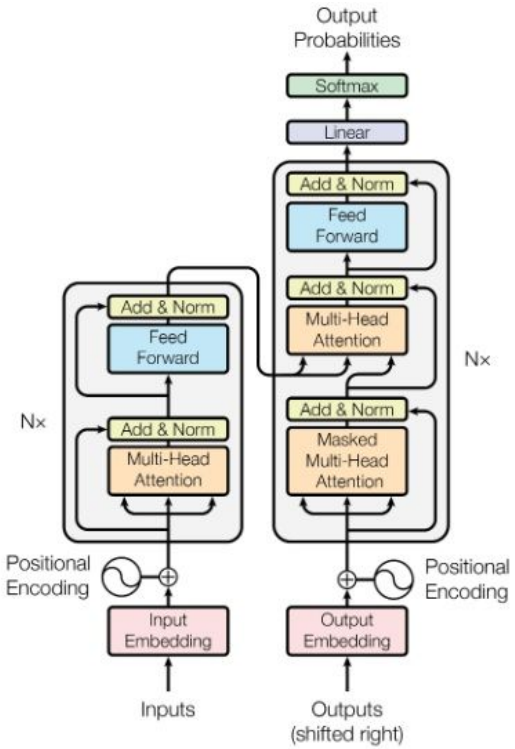
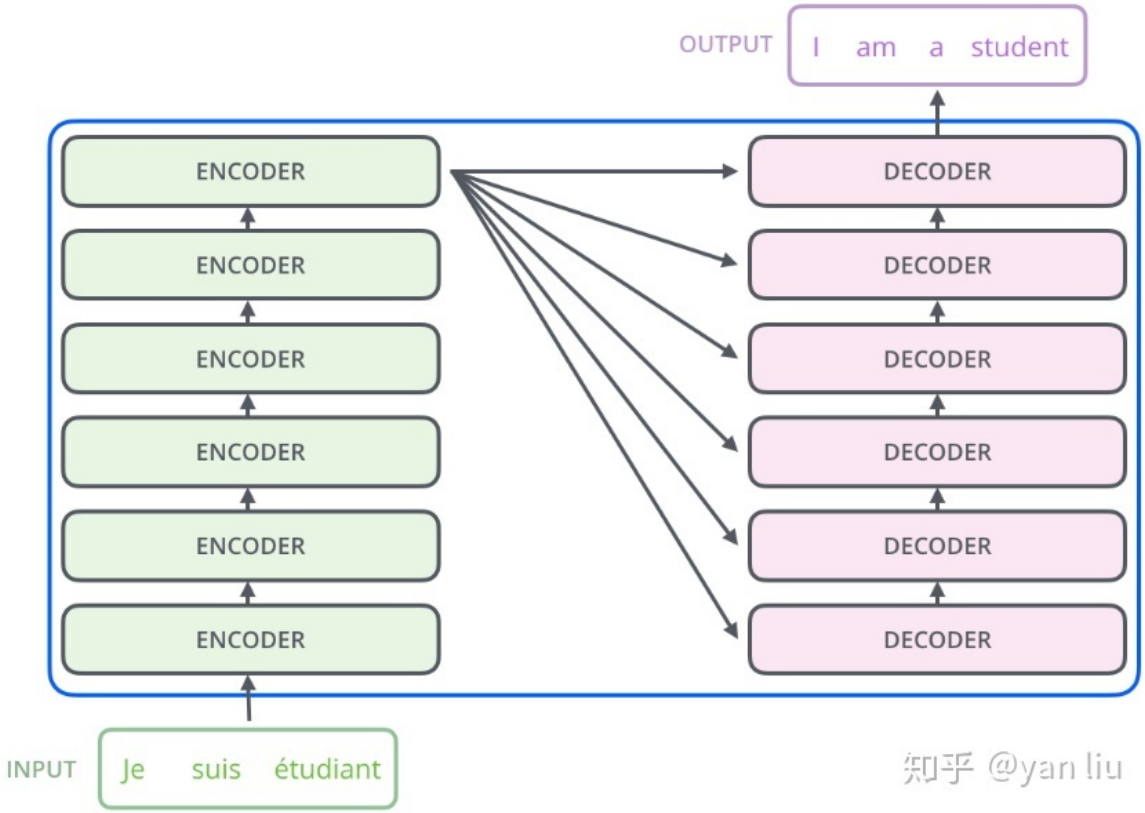


Figure 1: The Transformer - model architecture.

整体结构

Transformer

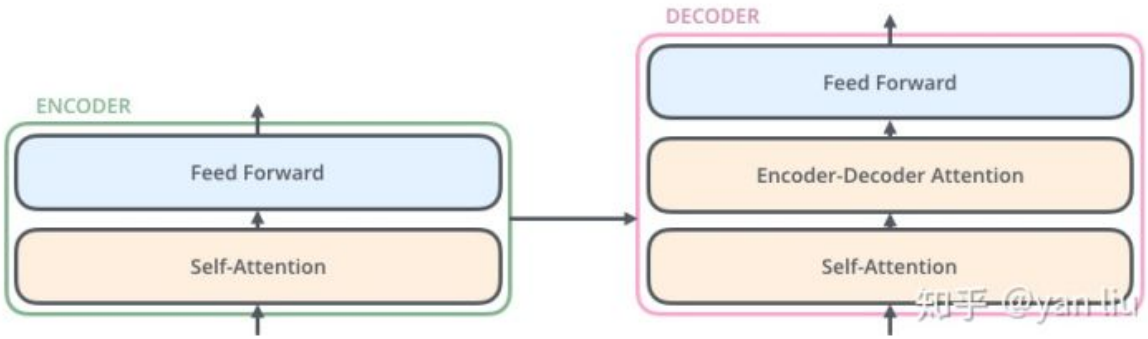
如论文中所设置的，编码器由 6 个编码 block 组成，同样解码器是 6 个解码 block 组成。与所有的生成模型相同的是，编码器的输出会作为解码器的输入。



Encoder and Decoder

Transformer

Decoder 和 encoder 的不同之处在于 Decoder 多了一个 Encoder-Decoder Attention，两个 Attention 分别用于计算输入和输出的权值。



1. Self-Attention：当前翻译和已经翻译的前文之间的关系。
2. Encoder-Decnoder Attention：当前翻译和编码的特征向量之间的关系。

Attention

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{23}$$

Feed Foward Network

$$FFN(Z) = \max(0, ZW_1 + b_1)W_2 + b_2 \tag{24}$$

Self Attention 的计算

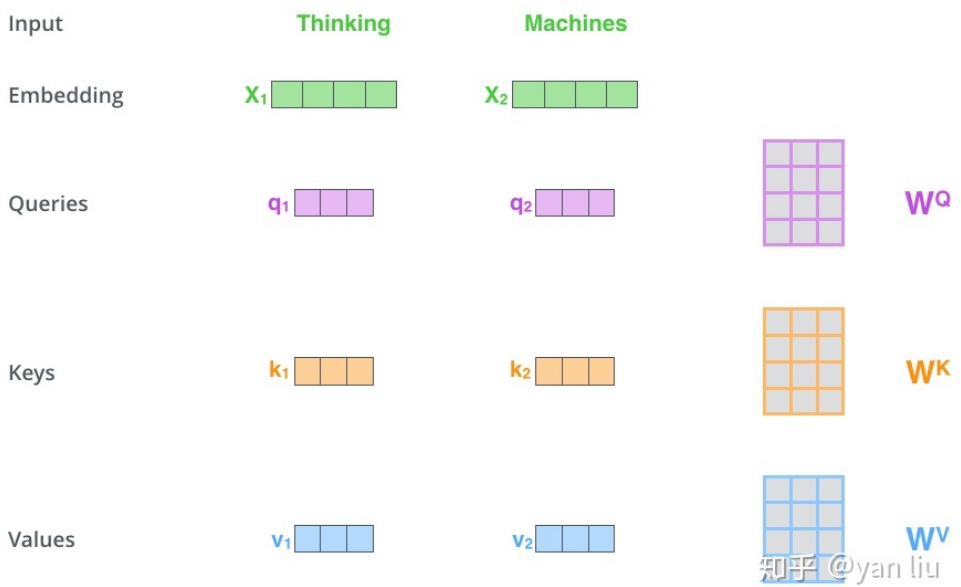
Transformer

1. 将输入单词转化成嵌入向量；
2. 根据嵌入向量得到 q ， k ， v 三个向量；
3. 为每个向量计算一个 score： score $= q \cdot k$ ；
4. 为了梯度的稳定，Transformer 使用了 score 归一化，即除以 $\sqrt{d_k}$ ；
5. 对 score 施以 softmax 激活函数；
6. softmax 点乘 Value 值 v ， 得到加权的每个输入向量的评分 v ；
7. 相加之后得到最终的输出结果 z ： $z = \sum v$ 。

Self Attention 的计算

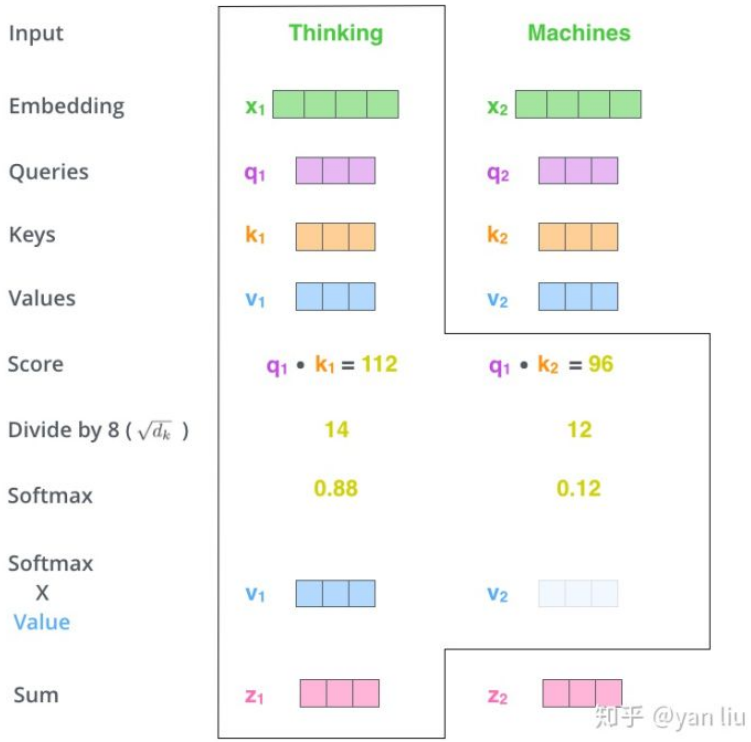
Transformer

在 self-attention 中，每个单词有 3 个不同的向量，它们分别是 Query 向量（Q），Key 向量（K）和 Value 向量（V），长度均是 64。它们是通过 3 个不同的权值矩阵由嵌入向量 X 乘以三个不同的权值矩阵 W^Q，W^K，W^V 得到，其中三个矩阵的尺寸也是相同的。均是 512×64。



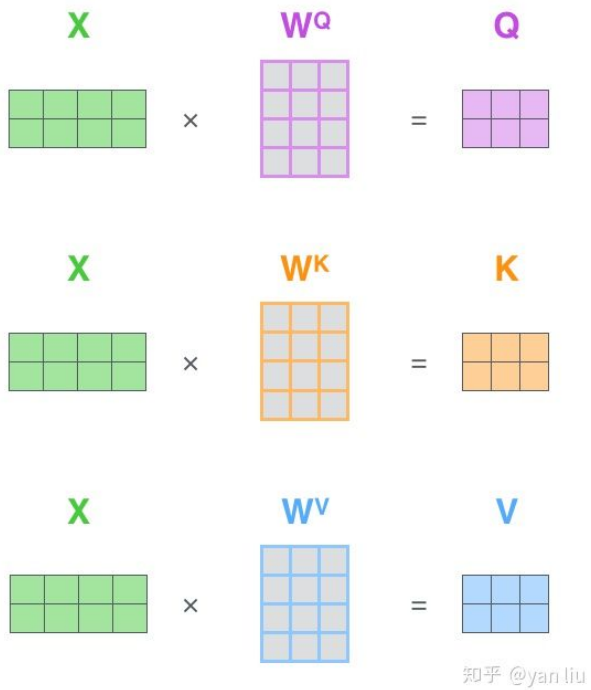
Self Attention 的计算

Transformer



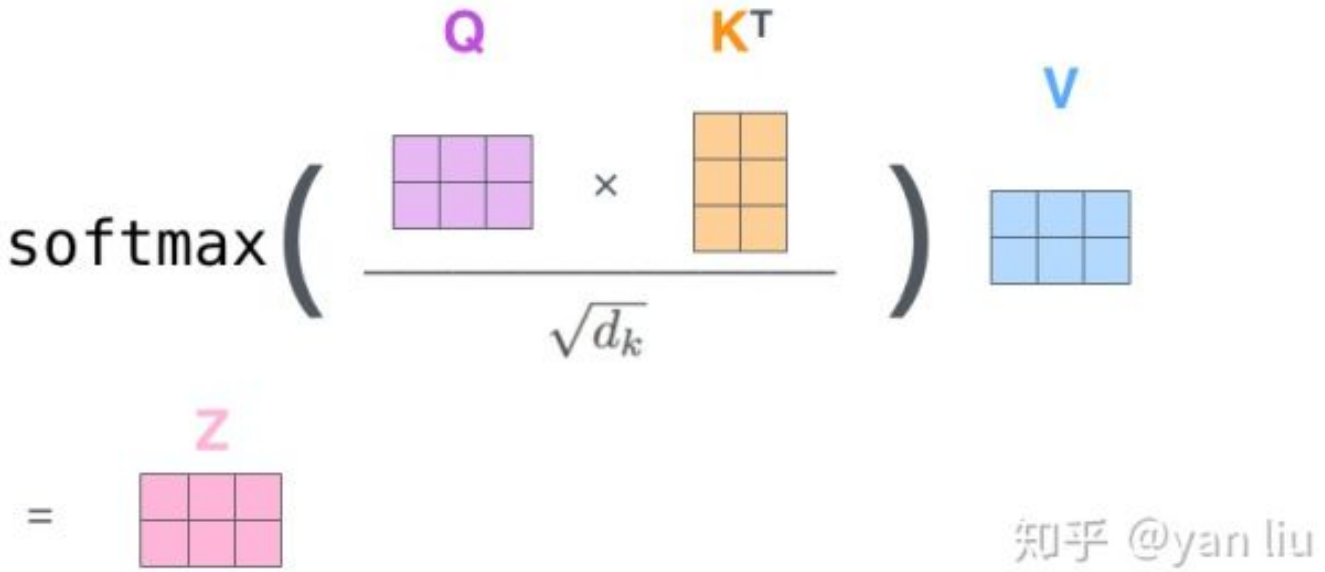
Self Attention 的矩阵形式计算

Transformer



Self Attention 的矩阵形式计算

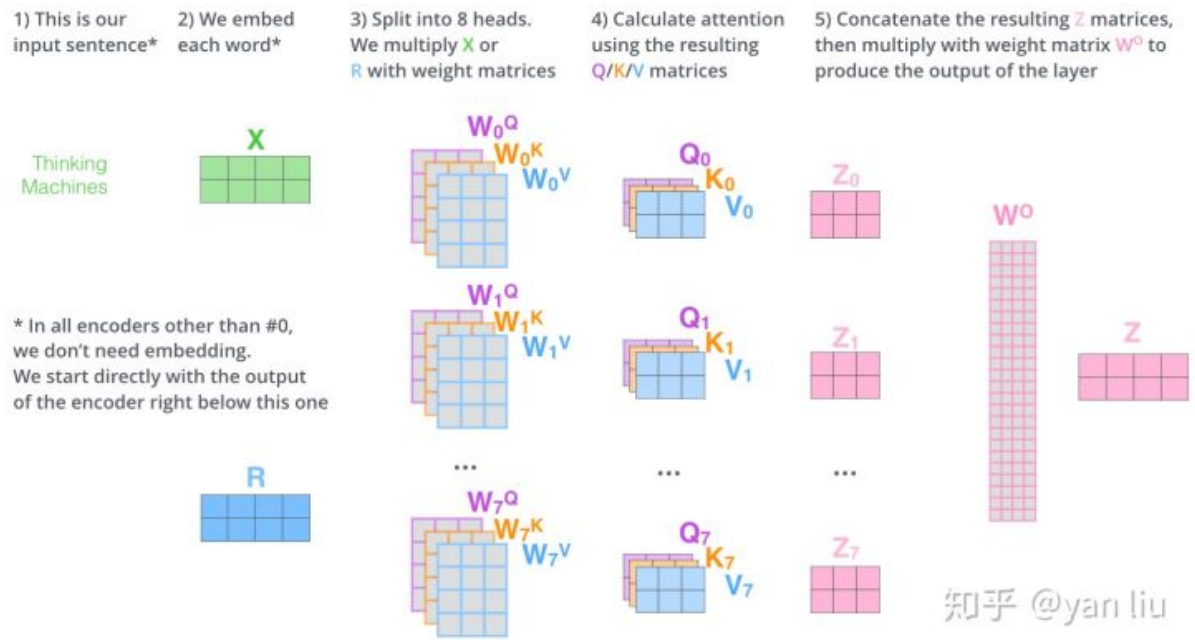
Transformer



Mult-head Attention 的计算

Transformer

Multi-Head Attention 相当于 h 个不同的 self-attention 的集成 (ensemble)，在这里我们以 h=8 举例说明。Multi-Head Attention 的输出分成 3 步：1. 将数据 X 分别输入到图 13 所示的 8 个 self-attention 中，得到 8 个加权后的特征矩阵 $Z_i, i \in \{1, 2, ..., 8\}$。2. 将 8 个 Z_i 按列拼成一个大的特征矩阵；3. 特征矩阵经过一层全连接后得到输出 Z 。



总结

Transformer

优点

(1) 虽然 Transformer 最终也没有逃脱传统学习的套路，Transformer 也只是一个全连接（或者是一维卷积）加 Attention 的结合体。但是其设计已经足够有创新，因为其抛弃了在 NLP 中最根本的 RNN 或者 CNN 并且取得了非常不错的效果，算法的设计非常精彩，值得每个深度学习的相关人员仔细研究和品位。(2) Transformer 的设计最大的带来性能提升的关键是将任意两个单词的距离是 1，这对解决 NLP 中棘手的长期依赖问题是非常有效的。(3) Transformer 不仅仅可以应用在 NLP 的机器翻译领域，甚至可以不局限于 NLP 领域，是非常有科研潜力的一个方向。(4) 算法的并行性非常好，符合目前的硬件（主要指 GPU）环境。

总结

Transformer

缺点

(1) 粗暴的抛弃 RNN 和 CNN 虽然非常炫技，但是它也使模型丧失了捕捉局部特征的能力，RNN + CNN + Transformer 的结合可能会带来更好的效果。(2) Transformer 失去的位置信息其实在 NLP 中非常重要，而论文中在特征向量中加入 Position Embedding 也只是一个权宜之计，并没有改变 Transformer 结构上的固有缺陷。