Language Model Survey

This paper introduces language models from Word2Vec to state-of-art models.

Take a look at the development of pre-trained language models since ELMo

模型	语言模型	特征抽取	上下文表征	最大亮点
ELMO	BiLM	BiLSTM	单向	2个单向语言模型拼接；
ULMFiT	LM	AWD-LSTM	单向	引入逐层解冻解决finetune中的灾难性问题；
SiATL	LM	LSTM	单向	引入逐层解冻+辅助LM解决finetune中的灾难性问题；
GPT1.0	LM	Transformer	单向	统一下游任务框架，验证Transformer在LM中的强大；
GPT2.0	LM	Transformer	单向	没有特定模型的精调流程，生成任务取得很好效果；
BERT	MLM	Transformer	双向	MLM获取上下文相关的双向特征表示；
MASS	LM+MLM	Transformer	单向/双向	改进BERT生成任务：统一为类似Seq2Seq的预训练框架；
UNILM	LM+MLM+S2SLM	Transformer	单向/双向	改进BERT生成任务：直接从mask矩阵的角度出发；
ENRIE1.0	MLM(BPE)	Transformer	双向	引入知识：3种[MASK]策略(BPE)预测短语和实体；
ENRIE	MLM+DEA	Transformer	双向	引入知识：将实体向量与文本表示融合；
MTDNN	MLM	Transformer	双向	引入多任务学习：在下游阶段；
ENRIE2.0	MLM+Multi-Task	Transformer	双向	引入多任务学习：在预训练阶段，连续增量学习；
SpanBERT	MLM+SPO	Transformer	双向	不需要按照边界信息进行mask；
RoBERTa	MLM	Transformer	双向	精细调参，舍弃NSP；
XLNet	PLM	Transformer-XL	双向	排列语言模型+双注意力流+Transformer

1. Word2vec

Word2vec is a "linear" language model. Since our goal is to learn word vectors, and the word vectors need to support some "linear" semantic operations semantically, a linear model is enough for the task, fast and elegant.

One of the essences of word2vec is to optimize the Softmax acceleration method of the language model by the way and replace the traditional hierarchical softmax and NCE methods with a seemingly open-minded "negative sampling" method. And what exactly is the "negative sampling" of this name?

Negative sampling

We know that for training a language model, the softmax layer is very difficult to calculate. After all, what you want to predict is which word is the current position, then the number of categories is equivalent to the size of the dictionary. Therefore, the number of categories of tens of thousands and hundreds of thousands are calculated as

softmax. Functions are, of course, laborious. However, if our goal is not to train an accurate language model, but only to train the by-products of the language model-word vectors, then we only need to use a "subtask" that is less expensive to calculate here.

Think about it. Is it arduous to give you 10,000 cards with numbers and let you find the larger ones? But if you extract the larger value in advance and mix it with five randomly drawn cards, letting you choose the larger value, is it easier?

Negative sampling is the idea, that is, instead of directly letting the model find the most likely word from the entire vocabulary, it is directly given this word (that is, the positive example) and several randomly sampled noise words (that is, the negative samples that are sampled) As long as the model can find the correct word from this, the goal is considered complete.
So the objective function corresponding to this idea is:

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

This idea of negative sampling was successfully applied to the BERT model, but the granularity changed from words to sentences.

char-level and context
Although there was a lot of work from 2015 to 2017 to try to start from char-level, to find a new way to get rid of the game rules of pre-trained word vectors, the actual measurement was only a short-lived and was quickly stunned [8] [9]. However, people also realize that char-level text also contains some patterns that are difficult to describe in word-level text. Therefore, on the one hand, the word vector FastText [5], which can learn char-level features, appears. In the supervised task, the aspect started to introduce char-level text representation through shallow CNN, HIghwayNet, RNN and other networks.

However, so far, word vectors are context-free. That is, the same word is always the same word vector in different contexts, which leads to the lack of word sense disambiguation (WSD) ability of the word vector model. Therefore, to make the word vector context-sensitive, people began to encode based on the word vector sequence in specific downstream tasks.

The most common encoding method is, of course, using RNN-based networks. Besides, there are also successful deep CNN encoding tasks (such as text classification [6], machine translation [7], and machine reading comprehension [4]). Of course! and! Google said, CNN is too vulgar, we want to use a fully connected network! (Crossed out) Self-attention! Then there was the Transformer model customized for NLP [11]. The transformer was proposed for machine translation tasks, but it also exerted great power in other fields such as retrieval dialog [3]. However, since it is found that there is a need for encoding in each NLP task, why not let the word vector have the context-dependent ability at the beginning? So there was ELMo [2].

ELMo

Of course, ELMo is not the first model to try to generate context-sensitive word vectors, but it is indeed a model that gives you good reason to abandon word2vec. After all, it is worth sacrificing the speed of point reasoning for a lot of hot performance. ELMo is a stacked bi-lstm on the model layer (strictly speaking, it trains two unidirectional stacked lstm), so of course, it has a good encoding ability. Training it is naturally a larger likelihood function of the language model standard, that is,

$$\sum_{k=1}^{N} \Big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \Big) .$$

However, the highlight of this ELMo is of course not the model layer, but it has indirectly demonstrated through experiments that the features learned in different layers in different layers of RNNs are different, so ELMo proposes to complete pre-training and migrate to downstream NLP In the task, a trainable parameter must be set for the original word vector layer and the hidden layer of each layer of RNN. These parameters are normalized by the softmax layer and multiplied to their corresponding layers and summed to play a weighting role. Then, the word vector obtained by the "weighted sum" is further scaled by a parameter to better adapt to the downstream task.

ps: In fact, this last parameter is still very important. For example, in word2vec, the variance of the word vector learned by cbow and sg is generally large. At this time, the variance and the word vector that matches the variance of the subsequent layers of the downstream task will converge faster. , Easier to have better performance

The mathematical expression is as follows

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

其中L=2就是ELMo论文中的设定，j=0时代表原始词向量层，j=1是lstm的第一隐层，j=2是第二隐层。s_j^{task}是参数被softmax归一化之后的结果（也就是说$s_0 + s_1 + \ldots + s_L = 1$）。

Through such a migration strategy, tasks that require word sense disambiguation are more likely to be given a large weight to the second hidden layer through training, while tasks that have obvious requirements for part-of-speech and syntax may have parameters for the first hidden layer. Learned relatively large values (experimental conclusions). In short, it is not surprising that a word vector with richer features that can be customized by downstream tasks is obtained. The effect is much better than word2vec.

But having said that, the goal of ELMo is to only learn context-sensitive and stronger word vectors, and its purpose is still to provide a solid foundation for downstream tasks.

And we know that just full and powerful encoding of the text (that is, getting very precise and rich features of each lexeme) is not enough to cover all NLP tasks. In QA,

machine reading comprehension (MRC), natural language reasoning (NLI), dialogue and other tasks, there are many more complex patterns to be captured, such as inter-sentence relationships. To this end, the network in the downstream task will add various fancy attentions.

With the need to capture more magical patterns, researchers have customized a variety of network structures for each downstream task, leading to the same model, which is hung up after a small change of task, even in the same task Significant performance loss will occur if the data set is replaced with another distribution, which does not conform to human language behaviors ~ We must know that human generalization ability is very strong, which shows that maybe the development trajectory of the entire NLP is wrong now, Especially under the leadership of SQuAD, exhausting various tricks and fancy structures to brush the list, what is the significance of NLP?

Fortunately, this road that has become more and more biased has finally been shut down by a model, which is Google's Bidirectional Encoder Representations from Transformers (BERT) [1].

BERT

The most important significance of this paper is not what model is used, nor how it is trained, but that it proposes a new type of game rules.

As said before, it is very unwise to deeply customize the complex model structure with poor generalization ability for each NLP task. Since ELMo will have such a big improvement over word2vec, this shows that the potential of pre-trained models goes far beyond providing an accurate word vector for downstream tasks, so can we directly pre-train a keel-level model? If it has fully described the characteristics of character-level, word-level, sentence-level, and even inter-sentence relationships, then in different NLP tasks, you only need to customize a very lightweight output layer (such as a single layer) for the task. MLP) is fine, after all, the model skeleton is already prepared.

And BERT did this thing, or in other words, it did it. As a general keel-level model, it easily challenged the deeply customized models on 11 tasks.
So how does it work?

Deep two-way encoding

First, it states that the previous pre-trained models for learning context-sensitive word vectors are not enough! Although in the downstream supervised tasks, the encoding method is already full of flamboyance, deep two-way encoding has become the standard for many complex downstream tasks (such as MRC, dialogue). But on the pre-trained model, the previous more advanced model is only based on the traditional language model, and the traditional language model is one-way (mathematically defined), that is,

$$p(s) = p(w0) \cdot p(w1|w0) \cdot p(w2|w1, w0) \cdot p(w3|w2, w1, w0) \ldots p(wn|context)$$

And it is often very shallow (think of the three layers of the LSTM stack, the train can't move, and various tricks are required), such as ELMo.
Also, although ELMo uses two-way RNNs for encoding, the RNNs in these two directions are trained separately, and only a simple addition is made at the loss layer at the end. This results in that for words in each direction, the words on the other side cannot be seen when it is encoded. The semantics of some words in a sentence will depend on some words on the left and right sides of it. Encoding from only one direction cannot be described clearly.
So why not do true two-way encoding like in downstream monitoring tasks?
The reason is clear at first thought. After all, traditional language models are designed to predict the next word. However, if the bidirectional encoding is done, it does not mean that the words to be predicted have been seen. Of course, such predictions are not Meaning. Therefore, in BERT, a new task is proposed to train a model that can be bidirectionally encoded in a supervised task. This task is called Masked Language Model (Masked LM).

Masked LM

As the name implies, Masked LM means that instead of giving a word that has already appeared like traditional LM to predict the next word, we directly make part of the whole sentence (randomly selected) to make it masked. This model If not, you can safely do two-way encoding, and then you can safely let the model predict what these covered words are. This task was called the cloze test (probably translated as a "gap test").

This causes some minor problems. Although the two-way encoding can be safely done in this way, the covered marks are also encoded in the encoding. These mask marks do not exist in downstream tasks. then what should we do? For this reason, to tune the model as much as possible to ignore the impact of these marks, the author tells the model "these are noises and noises! I can't trust it! Ignore them!"

80% probability to replace with "[mask]"
10% probability to replace a randomly sampled word
There is a 10% probability that no replacement will be made (although it is not replaced, it is still necessary to predict)

Encoder
In the selection of the encoder, the author did not use the bad street bi-lstm but used a deeper and better parallel transformer to do it. In this way, each word in the lexeme can directly encode each word in the sentence regardless of the direction and distance. On the other hand, I have a subjective feeling that the Transformer is easier to avoid the mask mark than lstm. After all, the self-attention process can completely weaken the mask mark in a targeted way, but how does the input gate in lstm treat the mask mark It is unknown.

As mentioned earlier, using the Transformer encoder directly will lose position information. Does the author here also have a sin and cos function encoding position like in the original Transformer paper? No, the author here simply and rudely trained a position embedding directly. That is to say, for example, if I truncate the sentence to a length of 50, then we have 50 positions, so 50 words are representing the position,

that is, from position 0 to position 49. Then give each position a randomly initialized word vector and let them train. Besides, in the combination of position embedding and word embedding, direct addition was selected in BERT.

Finally, in terms of depth, in the end, the full version of BERT's encoder was overwhelmingly superimposed with a 24-layer multi-head attention block. And each block contains 16 headers and 1024 units.

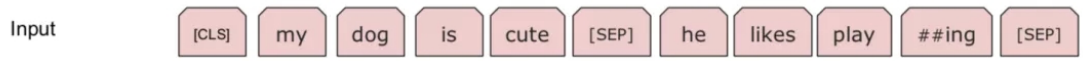
Learning sentence and sentence pair relationship expression
As mentioned before, in many tasks, encoding alone is not enough to complete the task (this only learns a bunch of token-level features), and it is necessary to capture some sentence-level patterns to complete SLI, QA, dialogue, etc. Sentence representation, interaction and matching tasks. In response, BERT has introduced another extremely important but extremely lightweight task to try to learn this model.

Sentence-level negative sampling
As mentioned in the previous chapter of word2vec, one of the essences of word2vec is to introduce an elegant negative sampling task to learn word-level representation. So what if we generalize this negative sampling process to a sentence-level? This is the key to BERT learning the sentence-level representation.

BERT is similar to word2vec here, but it constructs a sentence-level classification task. That is, a sentence is given first (equivalent to the given context in word2vec), its next sentence is a positive example (equivalent to the correct word in word2vec), and a sentence is randomly sampled as a negative example (equivalent to the random sampling in word2vec) Words), and then perform binary classification on the sentence-level (that is, determine whether the sentence is the next sentence or noise of the current sentence). Through this simple sentence-level negative sampling task, BERT can learn sentence representation as easily as word2vec learns word representation.

Sentence-level representation

BERT here is not like the common practice in downstream supervision tasks. Based on encoding, a global pooling is performed. It first starts with each sequence (for a sentence, it is two spelled sentences for the task. It is a sentence for other tasks) A special token is added in front of it, and it is marked as [CLS], as shown in the figure



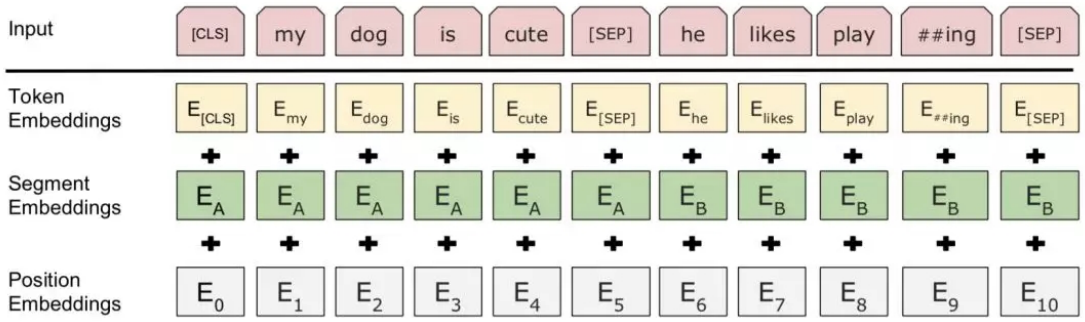
ps: [sep] is the delimiter between sentences. BERT also supports the representation of learning sentence pairs. Here [SEP] is to distinguish the cutting point of sentence pairs.

Then let the encoder perform deep encoding on [CLS], and the higher hidden layer of deep encoding is the representation of the entire sentence or sentence pair. This approach is a bit puzzling at first glance, but don't forget that Transformer can encode global information into each position regardless of space and distance, and [CLS] as a sentence or sentence pair representation is directly related to the output layer of the classifier Connected, so as a "gate" on the gradient backpropagation path, of course, I will find a way to learn the upper-level features related to classification.

Also, to allow the model to distinguish whether each word in it belongs to a "left sentence" or a "right sentence", the author introduces the concept of "segment embedding" to distinguish sentences. For sentence pairs, embedding A and embedding B are used to represent the left and right sentences respectively; for sentences, only embedding A is used. The embedding A and B are also trained with the model.

ps: This method feels as simple and rude as position embedding. It is very difficult to understand why BERT can still work on tasks that theoretically require the network to maintain symmetry.

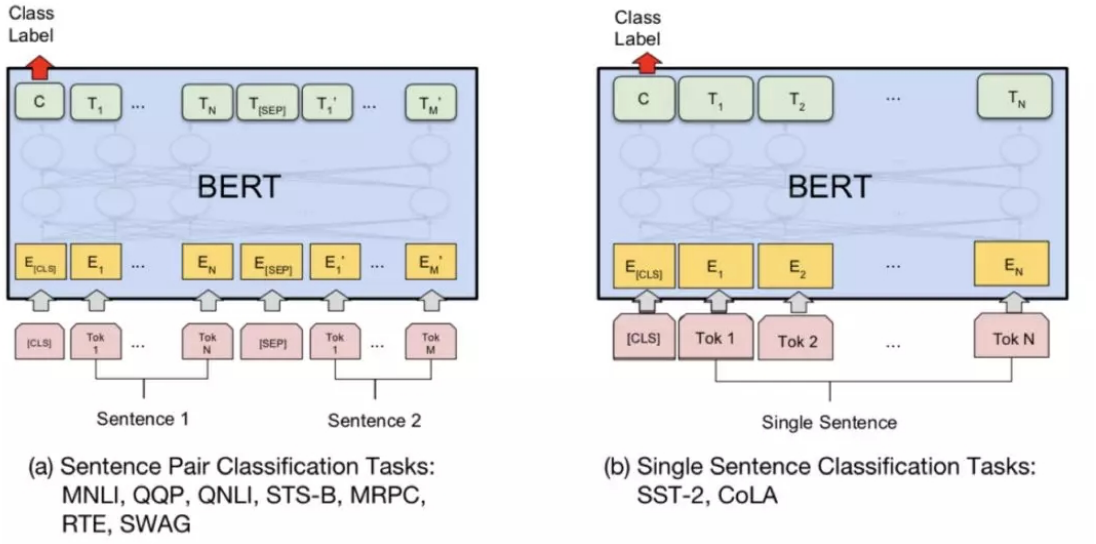
So, in the end, the representation of each token of BERT is made by adding the original word vector token embedding of the token, the position embedding mentioned above, and the segment embedding here, as shown in the figure:



Simple downstream task interface

It shows that the BERT model is a keel-level model rather than a word vector, that is, its interface design to various downstream tasks, or migration strategy.

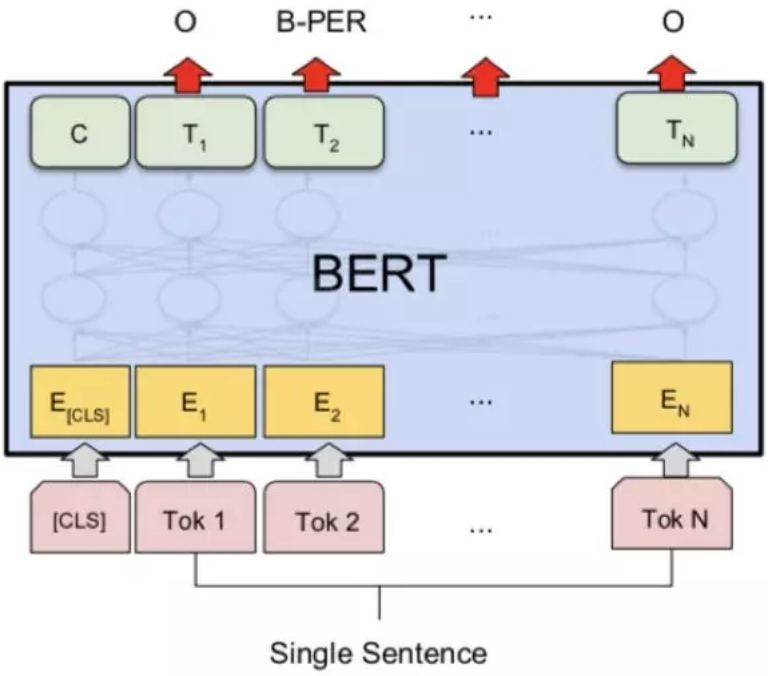
First, since the upper-level representations of sentences and sentence pairs are obtained, of course, for text classification tasks and text matching tasks (text matching is a text classification task, except that the input is a text pair), you only need to use the obtained The representation (ie the encoder outputs at the top level of the [CLS] lexemes) plus a layer of MLP is just fine.



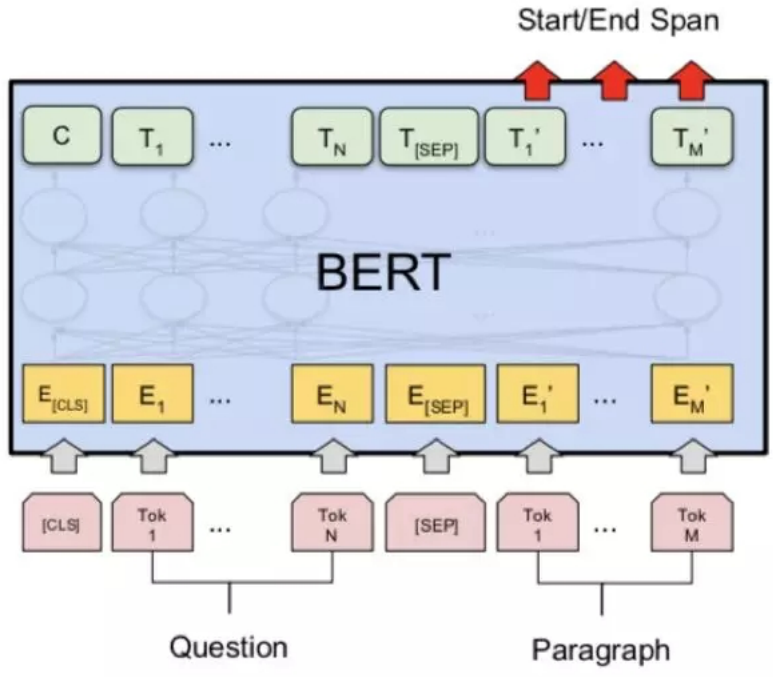
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

Now that the text has been encoded in both directions, it is only necessary to add a softmax output layer for the sequence labeling task, not even CRF.



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER



(c) Question Answering Tasks:
SQuAD v1.1

Here is the experiments' performance:

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

System	Dev EM	Dev F1	Test EM	Test F1
Leaderboard (Oct 8th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
#1 Single - nlnet	-	-	83.5	90.1
#2 Single - QANet	-	-	82.5	89.3
Published				
BiDAF+ELMo (Single)	-	85.8	-	-
R.M. Reader (Single)	78.9	86.3	79.5	86.6
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
$\text{BERT}_{\text{BASE}}$	96.4	92.4
$\text{BERT}_{\text{LARGE}}$	**96.6**	**92.8**

Table 3: CoNLL-2003 Named Entity Recognition results. The hyperparameters were selected using the Dev set, and the reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Table 4: SWAG Dev and Test accuracies. Test results were scored against the hidden labels by the SWAG authors. †Human performance is measure with 100 samples, as reported in the SWAG paper.

Here, I am going to discuss why Bert performs so good and introduce state-of-art language models that are derived from the Bert.

首先介绍 Transformer 的内核机制。

1）Multi-Head Attention 和 Scaled Dot-Product Attention

这个方法本质是 self-attention 通过 attention mask 动态编码变长序列，解决长距离依赖、无位置偏差、可并行计算

缩放点积，而不是点积模型?

当输入信息的维度 d 比较高，点积模型的值通常有比较大方差，从而导致 softmax 函数的梯度会比较小。因此，缩放点积模型可以较好地解决这一问题。

为什么是双线性点积模型（经过线性变换 Q K）？

双线性点积模型，引入非对称性，更具健壮性（Attention mask 对角元素值不一定是较大的，也就是说当前位置对自身的注意力得分不一定较高）。

相较于加性模型，点积模型具备哪些优点?

常用的 Attention 机制为加性模型和点积模型，理论上加性模型和点积模型的复杂度差不多，但是点积模型在实现上可以更好地利用矩阵乘积，从而计算效率更高（实际上，随着维度 d 的增大，加性模型会明显好于点积模型）。

多头机制为什么有效?

类似于 CNN 中通过多通道机制进行特征选择；

Transformer 中先通过切头（spilt）再分别进行 Scaled Dot-Product Attention，可以使进行点积计算的维度 d 不大（防止梯度消失），同时缩小 attention mask 矩阵。

2）Position-wise Feed-Forward Networks

FFN 将每个位置的 Multi-Head Attention 结果映射到一个更大维度的特征空间，然后使用 ReLU 引入非线性进行筛选，最后恢复回原始维度。

Transformer 在抛弃了 LSTM 结构后，FFN 中的 ReLU 成为了一个主要的提供非线性变换的单元。

3）Positional Encoding

将 Positional Embedding 改为 Positional Encoding，主要的区别在于 Positional Encoding 是用公式表达的、不可学习的，而 Positional Embedding 是可学习的（如 BERT），两种方案的训练速度和模型精度差异不大；但是 Positional Embedding 位置编码范围是固定的，而 Positional Encoding 编码范围是不受限制的。

BERT[13]为什么如此有效?

引入 Masked Language Model(MLM)预训练目标，能够获取上下文相关的双向特征表示；

引入 Next Sentence Prediction(NSP)预训练目标，擅长处理句子或段落的匹配任务；

引入强大的特征抽取机制 Transformer(多种机制并存)：

Multi-Head self attention：多头机制类似于“多通道”特征抽取，self attention 通过 attention mask 动态编码变长序列，解决长距离依赖（无位置偏差）、可并行计算；

Feed-forward ：在位置维度计算非线性层级特征；

Layer Norm & Residuals：加速训练，使“深度”网络更加健壮；

引入大规模、高质量的文本数据；

Q7：BERT 存在哪些优缺点?

优点：能够获取上下文相关的双向特征表示；

缺点：

生成任务表现不佳：预训练过程和生成过程的不一致，导致在生成任务上效果不佳；

采取独立性假设：没有考虑预测[MASK]之间的相关性，是对语言模型联合概率的有偏估计（不是密度估计）；

输入噪声[MASK]，造成预训练-精调两阶段之间的差异；

无法文档级别的 NLP 任务，只适合于句子和段落级别的任务；

Q8：BERT 擅长处理哪些下游 NLP 任务[14]?

1. 适合句子和段落级别的任务，不适用于文档级别的任务；

2. 适合处理高层语义信息提取的任务，对浅层语义信息提取的任务的提升效果不大（如一些简单的文本分类任务）；

3. 适合处理句子/段落的匹配任务；因此，在一些任务中可以构造辅助句（类似匹配任务）实现效果提升（如关系抽取/情感挖掘等任务）；

4. 不适合处理 NLG 任务；

Q9：BERT 基于“字输入”还是“词输入”好？（对于中文任务）

1. 如果基于“词输入”，会加剧 OOV 问题，会增大输入空间，需要利用大得多的语料去学习输入空间到标签空间的函数映射。

2. 随着 Transfomer 特征抽取能力，分词不再成为必要，词级别的特征学习可以纳入为内部特征进行表示学习。

Q10：BERT 为什么不适用于自然语言生成任务（NLG）？

1. 由于 BERT 本身在预训练过程和生成过程的不一致，并没有做生成任务的相应机制，导致在生成任务上效果不佳，不能直接应用于生成任务。

2. 如果将 BERT 或者 GPT 用于 Seq2Seq 的自然语言生成任务，可以分别进行预训练编码器和解码器，但是编码器–注意力–解码器结构没有被联合训练，BERT 和 GPT 在条件生成任务中只是次优效果。

五、BERT 系列模型进展介绍

这一部分介绍一些模型，它们均是对 BERT 原生模型在一些方向的改进。

Q11：针对 BERT 原生模型，后续的 BERT 系列模型是如何改进【生成任务】的?

1）MASS(微软)[15]

统一预训练框架:通过类似的 Seq2Seq 框架，在预训练阶段统一了 BERT 和 LM 模型；

Encoder 中理解 unmasked tokens；Decoder 中需要预测连续的

[mask]tokens，获取更多的语言信息；Decoder 从 Encoder 中抽取更多信

息；

当 k=1 或者 n 时，MASS 的概率形式分别和 BERT 中的 MLM 以及 GPT 中标

准的 LM 一致（k 为 mask 的连续片段长度））

2）UNILM (微软)[16]

统一预训练框架:和直接从 mask 矩阵的角度统一 BERT 和 LM；

3 个 Attention Mask 矩阵：LM、MLM、Seq2Seq LM；

注意：UNILM 中的 LM 并不是传统的 LM 模型，仍然是通过引入[MASK]实现

的；

Q12：针对 BERT 原生模型，后续的 BERT 系列模型是如何引入【知识】的?

1）ERNIE 1.0 (百度)[17]

在预训练阶段引入知识（实际是预先识别出的实体），引入 3 种[MASK]策略预测：

Basic-Level Masking：跟 BERT 一样，对 subword 进行 mask，无法获取高层次语义；

Phrase-Level Masking：mask 连续短语；

Entity-Level Masking：mask 实体；

2）ERNIE (THU)[18]

基于 BERT 预训练原生模型，将文本中的实体对齐到外部的知识图谱，并通过知识嵌入得到实体向量作为 ERNIE 的输入；

由于语言表征的预训练过程和知识表征过程有很大的不同，会产生两个独立的向量空间。为解决上述问题，在有实体输入的位置，将实体向量和文本表示通过非线性变换进行融合，以融合词汇、句法和知识信息；

引入改进的预训练目标 Denoising entity auto-encoder (DEA)：要求模型能够根据给定的实体序列和文本序列来预测对应的实体；

Q13：针对 BERT 原生模型，后续的 BERT 系列模型是如何引入【多任务学习机制】的?

多任务学习(Multi-task Learning)[19]是指同时学习多个相关任务，让这些任务在学习过程中共享知识，利用多个任务之间的相关性来改进模型在每个任务的性能和泛化能力。多任务学习可以看作是一种归纳迁移学习，即通过利用包含在相关任务中的信息作为归纳偏置(Inductive Bias)来提高泛化能力。多任务学习的训练机制分为同时训练和交替训练。

1）MTDNN(微软)[20]：在下游任务中引入多任务学习机制

ERNIE 2.0 (百度)[21]：在预训练阶段引入多任务学习

MTDNN 是在下游任务引入多任务机制的，而 ERNIE 2.0 是在预训练引入多任务学习（与先验知识库进行交互），使模型能够从不同的任务中学到更多的语言知识。

主要包含 3 个方面的任务：

word-aware 任务：捕捉词汇层面的信息；

structure-aware 任务：捕捉句法层面的信息；

semantic-aware 任务：捕捉语义方面的信息；

主要的方式是构建增量学习（后续可以不断引入更多的任务）模型，通过多任务学习持续更新预训练模型，这种连续交替的学习范式不会使模型忘记之前学到的语言知识。

将 3 大类任务的若干个子任务一起用于训练，引入新的任务时会将继续引入之前的任务，防止忘记之前已经学到的知识，具体是一个逐渐增加任务数量的过程[22]：

(task1)->(task1,task2)->(task1,task2,task3)->…->(task1,

task2,…,taskN)，

Q14：针对 BERT 原生模型，后续的 BERT 系列模型是如何改进【mask 策略】的?

原生 BERT 模型：按照 subword 维度进行 mask，然后进行预测；

BERT WWM(Google)：按照 whole word 维度进行 mask，然后进行预测；

ERNIE 等系列：引入外部知识，按照 entity 维度进行 mask，然后进行预测；

SpanBert：不需要按照先验的词/实体/短语等边界信息进行 mask，而是采取随机 mask：

采用 Span Masking：根据几何分布，随机选择一段空间长度，之后再根据均匀分布随机选择起始位置，最后按照长度 mask；通过采样，平均被遮盖长度是 3.8 个词的长度；

引入 Span Boundary Objective：新的预训练目标旨在使被 mask 的 Span 边界的词向量能学习到 Span 中被 mask 的部分；新的预训练目标和 MLM 一起使用；

注意：BERT WWM、ERNIE 等系列、SpanBERT 旨在隐式地学习预测词（mask 部分本身的强相关性）之间的关系[23]，而在 XLNet 中，是通过 PLM 加上自回归方式来显式地学习预测词之间关系；

Q15：针对 BERT 原生模型，后续的 BERT 系列模型是如何进行【精细调参】的?

RoBERTa(FaceBook)[24]

丢弃 NSP，效果更好；

动态改变 mask 策略，把数据复制 10 份，然后统一进行随机 mask；

对学习率的峰值和 warm-up 更新步数作出调整；

在更长的序列上训练：不对序列进行截短，使用全长度序列；

六、XLNet 的内核机制探究

在 BERT 系列模型后，Google 发布的 XLNet 在问答、文本分类、自然语言理解等任务上都大幅超越 BERT；XLNet 的提出是对标准语言模型（自回归）的一个复兴[25]，提出一个框架来连接语言建模方法和预训练方法。

Q16：XLNet[26]提出的背景是怎样的?

对于 ELMO、GPT 等预训练模型都是基于传统的语言模型（自回归语言模型 AR），自回归语言模型天然适合处理生成任务，但是无法对双向上下文进行表征，因此人们反而转向自编码思想的研究（如 BERT 系列模型）；

自编码语言模型（AE）虽然可以实现双向上下文进行表征，但是：

BERT 系列模型引入独立性假设，没有考虑预测[MASK]之间的相关性；

MLM 预训练目标的设置造成预训练过程和生成过程不一致；

预训练时的[MASK]噪声在 finetune 阶段不会出现，造成两阶段不匹配问题；

有什么办法能构建一个模型使得同时具有 AR 和 AE 的优点并且没有它们缺点呢?

Q17：XLNet 为何如此有效：内核机制分析

1）排列语言模型（Permutation LM，PLM）

如果衡量序列中被建模的依赖关系的数量，标准的 LM 可以达到上界，不像 MLM 一样，LM 不依赖于任何独立假设。借鉴 NADE[27]的思想，XLNet 将标准的 LM 推广到 PLM。

为什么 PLM 可以实现双向上下文的建模?

PLM 的本质就是 LM 联合概率的多种分解机制的体现；

将 LM 的顺序拆解推广到随机拆解，但是需要保留每个词的原始位置信息

（PLM 只是语言模型建模方式的因式分解/排列，并不是词的位置信息的重新排列！）

如果遍历 $T!$ 种分解方法，并且模型参数是共享的，PLM 就一定可以学习到各种双向上下文；换句话说，当我们把所有可能的$T!$ 排列都考虑到的时候，对于预测词的所有上下文就都可以学习到了！

由于遍历 $T!$ 种路径计算量非常大（对于 10 个词的句子，10!=3628800）。因此实际只能随机的采样$T!$里的部分排列，并求期望；

2）Two-Stream Self-Attention

如果采取标准的 Transformer 来建模 PLM，会出现没有目标(target)位置信息的问题。问题的关键是模型并不知道要预测的到底是哪个位置的词，从而导致具有部分排列下的 PLM 在预测不同目标词时的概率是相同的

怎么解决没有目标(target)位置信息的问题?

对于没有目标位置信息的问题，XLNet 引入了 Two-Stream Self-Attention：

Query 流就为了预测当前词，只包含位置信息，不包含词的内容信息；

Content 流主要为 Query 流提供其它词的内容向量，包含位置信息和内容信息；

3） 融入 Transformer-XL 的优点（具体见 Q18）

Q18：Transformer-XL[28]怎么实现对长文本建模?

BERT(Transformer)的较大输入长度为 512，那么怎么对文档级别的文本建模?

vanilla model 进行 Segment，但是会存在上下文碎片化的问题（无法对连续文档的语义信息进行建模），同时推断时需要重复计算，因此推断速度会很慢；

Transformer-XL 改进

对于每一个 segment 都应该具有不同的位置编码，因此 Transformer-XL 采取了相对位置编码；

前一个 segment 计算的 representation 被修复并缓存，以便在模型处理下一

个新的 segment 时作为扩展上下文 resume；

较大可能依赖关系长度增加了 N 倍，其中 N 表示网络的深度；

解决了上下文碎片问题，为新段前面的 token 提供了必要的上下文；

由于不需要重复计算，Transformer–XL 在语言建模任务的评估期间比 vanilla

Transformer 快 1800+倍；

引入 recurrence mechanism(不采用 BPTT 方式求导)：

引入相对位置编码方案：

七、预训练语言模型的未来

上述的【预训练语言模型】主要从 2 大方面进行介绍：一是总的对比；二是分

别介绍单向语言模型、BERT 系列模型、XLNet 模型。

可以看出，未来【预训练语言模型】更多的探索方向主要为[25]：

复兴语言模型：进一步改进语言模型目标，不断突破模型的上界；

大数据、大算力：将大数据、大算力推到极致；

更快的推断：轻量级模型是否有可能达到 SOTA 效果?

引入更丰富的知识信息，更精细的调参，更有价值的 MASK 策略；

统一条件生成任务框架，如基于 XLNet 统一编码和解码任务，同时可考虑更快的解码方式；

[1] 2018 | BERT- Pre-training of Deep Bidirectional Transformers for Language Understanding
[2] 2018NAACL | Deep contextualized word representations
[3] 2018 ACL | Multi-Turn Response Selection for Chatbots with Deep Attention Matching Network
[4] 2018ICLR | Fast and Accurate Reading Comprehension by Combining Self-Attention and Convolution
[5] 2017TACL | Enriching Word Vectors with Subword Information
[6] 2017ACL | Deep Pyramid Convolutional Neural Networks for Text Categorization
[7] 2017 | Convolutional Sequence to Sequence Learning
[8] 2017 | Do Convolutional Networks need to be Deep for Text Classification ?
[9] 2016 | Convolutional Neural Networks for Text Categorization/ Shallow Word-level vs. Deep Character-level
[10] 2013NIPS | Distributed-representations-of-words-and-phrases-and-their-compositionality