Methods of obtaining sentence representation

1. Statistical Language Model

The essence of language model is to predict the probability of a natural language text, that is, if the text is represented by Si, then language model requires P (SI) If we calculate this probability according to the idea of infinite approximation of frequency to probability in the law of large numbers, we naturally need to use this text in all text sets produced in all human history to calculate the frequency P (SI) of this text first, and then we can get it through the following formula:

$$P(S_i) = \frac{c(S_i)}{\sum_{j=0}^{\infty} c(S_j)}$$

This formula is simple enough, but the problem is that the statistics of all the historical corpora of the whole mankind can't be realized obviously. So in order to make this impossible statistical task possible, some people don't make the text as a whole, but break it up into words. Through the probability relationship between each word, we can get the probability of the whole text. Suppose the sentence length is t and the word is x, that is:

$$P(S_i) = P(x_0, x_1, \ldots, x_T) = P(x_0)P(x_1|x_0)P(x_2|x_0, x_1)\ldots P(x_T|x_0, x_1, x_2, \ldots, x_{T-1})$$

However, the calculation of this formula is still too complex. We usually introduce Markov hypothesis: assume that a word in a sentence is only related to the n words in front of it, especially when n = 1, the probability calculation formula of the sentence is the most concise:

$$P(S_i) = P(x_0, x_1, \ldots, x_T) = P(x_0)P(x_1|x_0)P(x_2|x_1)\ldots P(x_T|x_{T-1}) = P(x_0)\Pi_{i=0}^{T-1}P(x_{i+1}|x_i)$$

And the statistics of word frequency is used to estimate the conditional probability in this language model, as follows:

$$P(x_{i+1}|x_i) = \frac{c(x_{i+1}, x_i)}{c(x_i)}$$

In this way, the calculation of language model is finally feasible. However, there are many problems in this statistical language model:

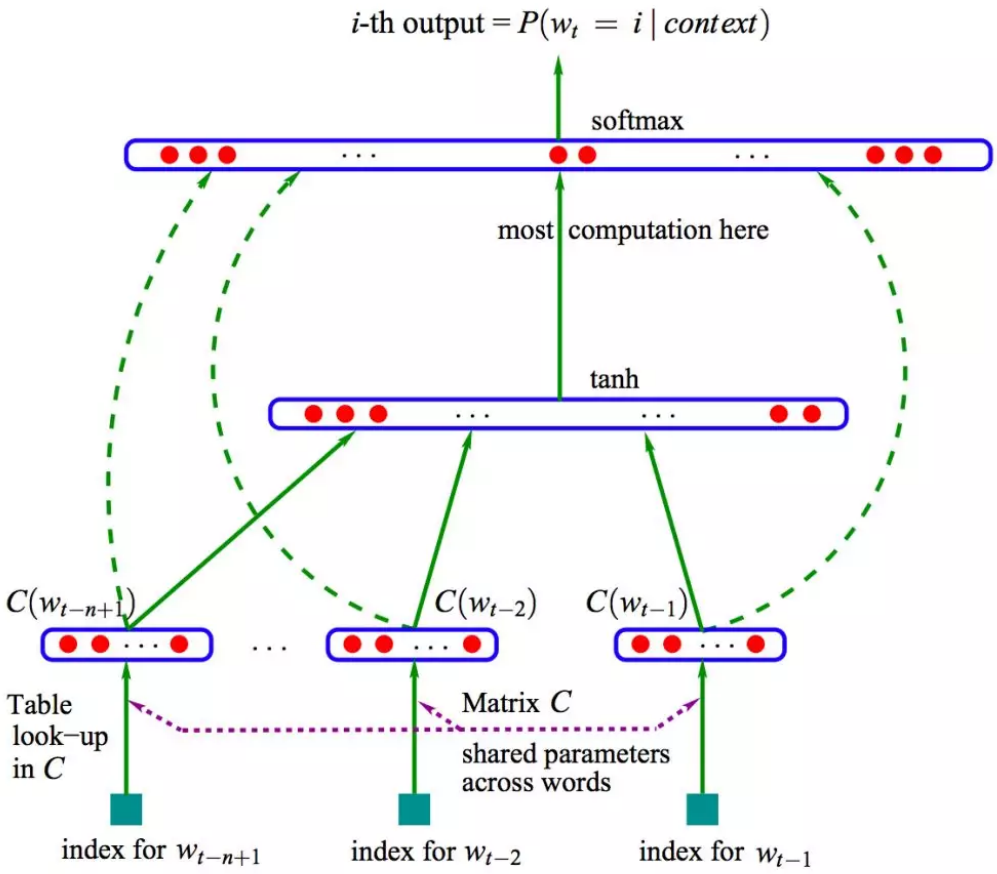
Firstly, in many cases, there are many zeros when calculating $c(x_{i+1}, x_i)$, especially when the value of n is large, the phenomenon of zero caused by data sparsity becomes very serious. So a very important direction of statistical language model is to design various smoothing methods to deal with this situation.

Secondly, another more serious problem is that the language model based on statistics can't make n very big. Generally speaking, it's common in 3-gram. If it's bigger, the computational complexity will increase exponentially. Because of this problem, the statistical language model cannot model the long context dependency in the language.

Thirdly, statistical language model cannot represent the similarity between words.

2. NNLM

The existence of these shortcomings forced bengio to integrate the idea of deep learning into the language model for the first time in his classic paper a neural probibilistic language model in 2003, and found the nnlm (neural net language) obtained by training When the first level parameters of the model are used as the distributed representation of words, the similarity between words can be obtained well.



The log likelihood function of the maximum objective function of nnlm, excluding the regularization term, is essentially an n-gram language model, as follows:

$$L = \frac{1}{T}\sum_t \log P(w_t|w_{t-1},\ldots,w_{t-n+1};\theta) + R(\theta)$$

$$P(w_t|w_{t-1},\ldots,w_{t-n+1};\theta) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

Here, the network structure of nnlm can be divided into three parts: the first part, mapping from word to word vector, mapping through C matrix, the number of parameters is | V | * m; the second part, mapping from X to hidden layer, mapping through matrix H, the number of parameters here is | h | * m * (n-1); the third part, mapping from hidden layer to output layer, mapping through matrix U. The number of parameters is | v| * |; h|; the fourth part, mapping from X to output layer, through matrix W, the number of parameters is | * m * (n-1).

Therefore, if the number of parameters of the offset term (where the output layer is|v|, and the input layer to the hidden layer is|h|), the number of parameters of nnlm is:

$$|V|(1+|H|+mn)+|H|(1+mn-m)$$

It can be seen that the number of nnlm parameters is a linear function of the window size n, which enables nnlm to model longer dependencies. However, the main contribution of nnlm is to use the first level feature mapping matrix of the model as a distributed representation of words, so that a word can be represented as a vector form, which directly inspired the work of word2vec later.

The biggest disadvantage of nnlm is that it has many parameters and slow training. In addition, the input of nnlm is required to be fixed length NNN, which is not flexible in itself and can not use complete historical information.

3. Word2Vec - CBOW

Word 2vec for the optimization of predecessors, mainly in two aspects: model simplification and training skills optimization. Let's take a look at the simplification of the model, which is known as cbow and skip gram.
For cbow, we can see from its name. Its full name is continuous bag of words, that is, continuous bag model. Why take this name? Let's look at its objective function first.

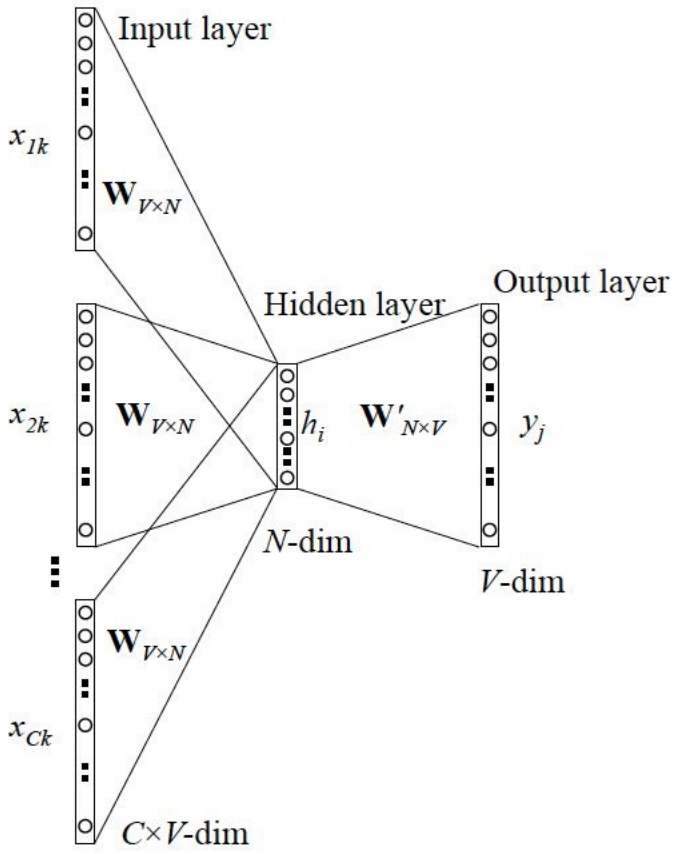
$$\frac{1}{T}\sum_{t=1}^{T}\log P(w_t|c_t)$$

$$P(w_t|c_t)=\frac{exp(e'(w_t)^T x)}{\sum_{i=1}^{|V|} exp(e'(w_i)^T x)}, x=\sum_{i\in c} e(w_i)$$

First of all, cbow has no hidden layer, but only two-tier structure in essence. The input layer makes the target context C Each word vector in is simply summed (of course, it can also be averaged) to get the context vector, and then directly calculate the point product with the output vector of the target word. The objective function is to make this point product with the target word vector get the maximum value, and the corresponding point product with the non target word get the minimum value as far as possible.

It can be seen from this that the first feature of cbow is to cancel the hidden layer in nnlm and directly connect the input layer and the output layer; the second feature is that the word order in the context has been discarded (this is continuous in the name) when the context vector is calculated Third, because the final objective function is still the objective function of the language model, we need to traverse every word in the corpus in order (this is the source of bag of words in the name).

So with these characteristics (especially the second and third points), mikolov named this simple model cbow, a simple but effective model.

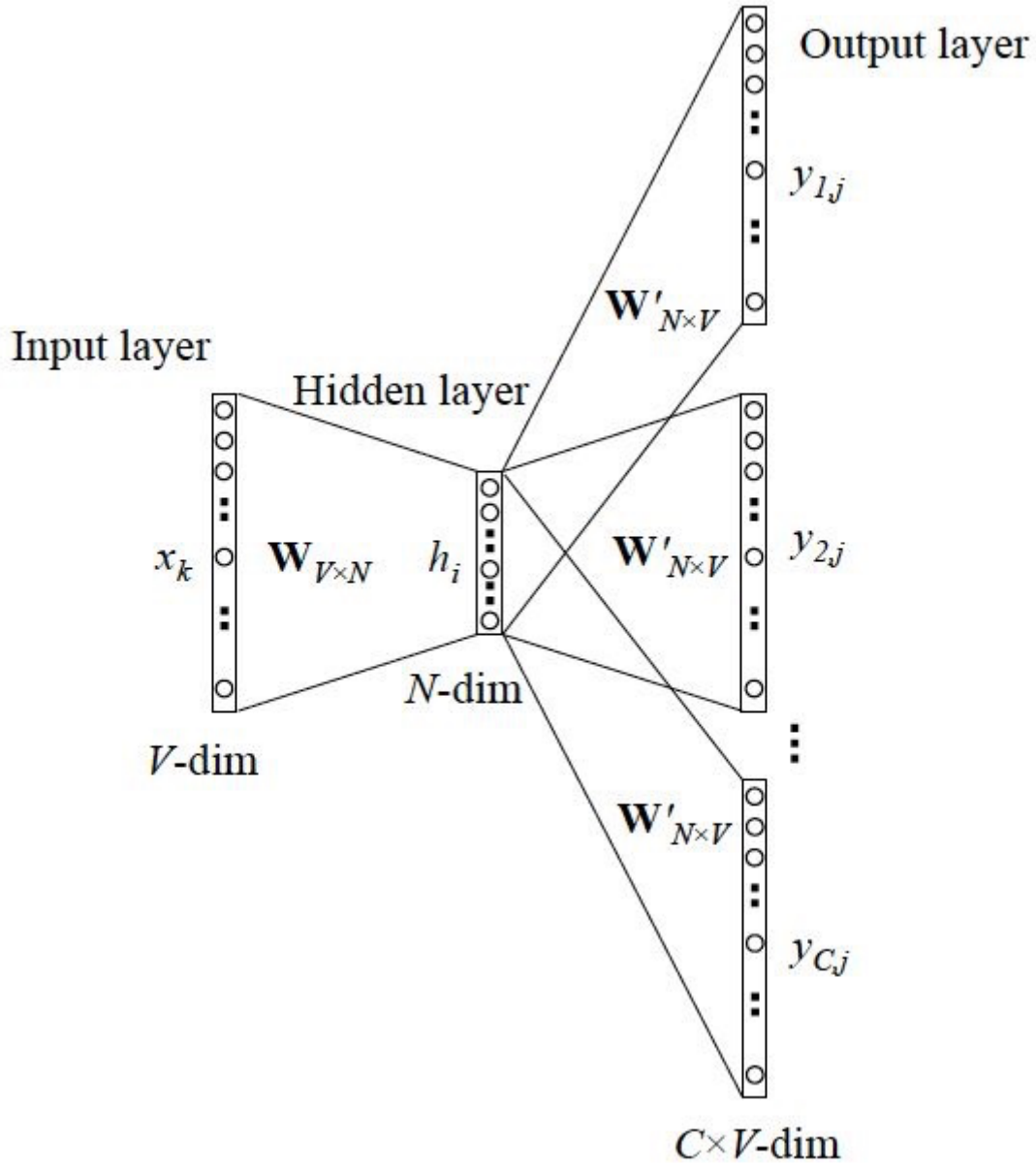


4. Word2Vec - Skip-gram

Corresponding to cbow, the basic idea of skip gram model is very similar to cbow, but in a different direction: cbow is to let the output vector E '(WT) of the target word fit the vector X of the context; while skip gram is to let the output vector of each word in the context fit the vector E (WT) of the current input word as much as possible, opposite to the direction of cbow, so its objective function is as follows:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in c}\log P(w_j|w_t)$$

$$P(w_j|w_t)=\frac{exp(e'(w_j)^T e(w_t))}{\sum_{i=1}^{|V|} exp(e'(w_i)^T e(w_t))}$$



In fact, no matter cbow or skip gram, they are essentially two fully connected layers, and there is no other layer in between. It has nothing to do with the number of words selected in the context, that is to say, it finally avoids the following order n in n-gram The problem of increasing computational complexity.

The method of reducing the complexity of calculation again：

a. Hierachical Softmax

The basic idea of hierachical softmax is to first construct a Huffman tree according to the word frequency of each word in the dictionary, so as to ensure that the words with larger word frequency are in the

relatively shallow layer, and the words with lower word frequency are in the deeper leaf node of the Huffman tree, and each word is in a certain leaf node of the Huffman tree.

Second, the original classification problem of a | V | is transformed into a log | V | sub classification problem. To put it simply, when the original P (WT | CT) is to be calculated, because the common softmax is used, the probability of each word in the dictionary is bound to be required.

In order to reduce the amount of calculation in this step, in Hierachical softmax, the probability of the current word WT in its context is also calculated. Just turn it into a path prediction problem in the Huffman tree, because the current word WT is in Huffman tree The tree corresponds to a path, which is composed of the root node in the binary tree, a series of intermediate parent nodes, and finally the leaf node of the current word. Then on each parent node, it corresponds to a binary classification problem (essentially an LR classifier), and the Huffman tree construction process ensures that the tree depth is log |V|, so we only need to do log|v| secondary classification to get the size of P (wt|ct), which has greatly reduced the amount of calculation compared with the original|v|.

b. Down sampling

Common softmax has too much computation because it treats all the other non target words in the dictionary as negative examples, and the idea of negative sampling is particularly simple, that is, every time some words are randomly sampled according to a certain probability as negative examples, so it only needs to calculate these negative samples

5. GloVe

Let's first look at glove's loss function

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Among them, Xij is the co occurrence frequency of two words I and j in a certain window size (however, glove has made some improvements on it, and the co occurrence frequency has a corresponding attenuation coefficient, so that the farther the words are, the smaller the co occurrence frequency is); f (Xij) is a weight coefficient, and the main purpose is to co-occurrence more pairs The contribution to the objective function should be larger, but it can not be increased without limitation, so the maximum value of pair with too large co-occurrence frequency should be limited to prevent the whole objective function from being dominated by the pair with too large frequency during training.

The rest is the core part of the algorithm. The two b values are two bias terms. Leaving aside, the rest $(w_i^T w_j - \log X_{ij})^2$ is actually a common mean square error function. Wi is the vector of the current word. WJ corresponds to the word vector of the co-occurrence words in the same window. The vector dot multiplication of the two should try to fit the logarithm of their co-occurrence frequency.

Intuitively, if the frequency of co occurrence of two words is higher, the value of the pair is certainly higher, so the algorithm requires that the point multiplication of the two word vectors is larger, and the point multiplication of the two word vectors is larger, which actually contains two meanings:

First, the larger the modulus of each word vector is required. Generally speaking, it is intuitive to remove words with very high frequency (such as stop words). For words with clear semantics, the length of their word vector modulus will increase with the increase of word frequency. Therefore, the greater the frequency of co

occurrence of two words, the greater the modulus of each word vector is required Second, it is intuitive to ask that the smaller the angle between the two word vectors is, because the greater the frequency in the same context, the closer the semantics of the two words are, the smaller the angle between the word vectors is.

In addition, it can be found from the loss function used by glove that its training mainly consists of two steps: statistical co-occurrence matrix and training to obtain word vector, which is actually not the model we usually understand.

6. fastText

Both word2vec and glove do not need manual labeled surveillance data. They only need surveillance signals in the language to complete the training. In contrast, fasttext uses text classification data with supervision marks to complete the training.

In essence, there is nothing special. The model framework is cbow, but it is different from the common cbow in two aspects: input data and prediction target.

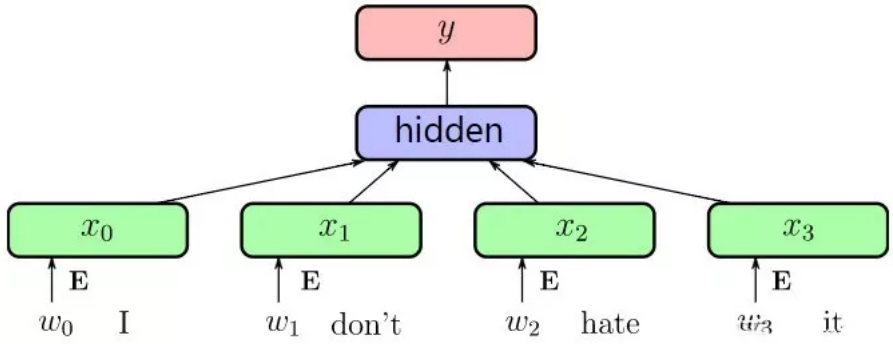
In the input data, cbow inputs the vector sum or average of all the other words except the target words in an interval. In order to use more word order information, fasttext turns bag of words into Bag of features, that is, the input X in the figure below is no longer just a word, but also the information of bigram or trigram.

The second difference is that the cbow prediction target is a word in the context, and the fasttext prediction target is the category of the current input text. Just because this text category is needed, fasttext is a supervision model.

The same thing is that the network structure of fasttext is basically the same as that of cbow, and the hierachical softmax technique is also used in the classification of output layer to speed up the training.

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log f(BAx_n), x_n = \sum_{i=1}^{l_n} x_{n,i}$$

Here $x_{n,i}$ is the I word of the nth document i n the corpus and the characteristic information of n-gram. From this loss function, we can know that fasttext also has only two fully connected layers, a and B, where a is the final word vector information that can be obtained.



Fasttext is characterized by its fast speed, which is also verified by detailed experiments in this paper. On some classified data sets, fasttext can generally reduce the time of CNN structure model to a few hours or even days, which is only a few seconds.

7. PV-DM

The full name of pv-dm is distributed memory model of paragraph vectors, which is similar to cbow. It also predicts the next word through context. However, in the input layer, it also maintains a look-up of document ID mapped to a vector Table, the purpose of the model is to input the current document vector and context vector into the model, and let the model predict the next word.

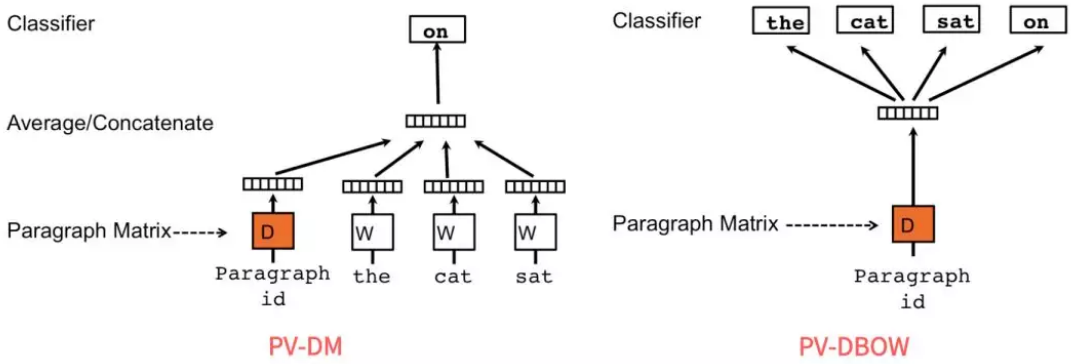
After the training, for the existing document, you can quickly get the vector of the document by looking up the table directly. For a new document, you need to add the corresponding column to the existing look-up table, and then go through the

training process again, but fix the other parameters at this time, and only adjust the look-up Table, after convergence, you can get the vector corresponding to the new document.

8. PV-DBOW

The full name of pv-dbow is distributed bag of words version of paragraph vector. Similar to skip gram, it uses documents to predict words in documents. During training, it randomly samples some text segments, and then samples a word from this segment to let pv-dbow model predict the word.

In essence, it is the same as skip gram. This method has a fatal weakness, that is, in order to obtain the vector of new documents, we have to continue to go through the training process, and because the model is mainly aimed at the process of document vector predicting word vector, in fact, it is difficult to represent the richer semantic structure between words, so these two methods of obtaining document vector have not been widely used.

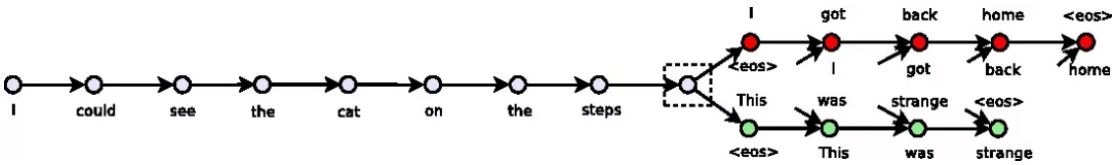


9. Skip-thoughts

The idea of skip gram is also used for reference, but unlike the method of using documents to predict words in pv-dbow, skip ideas directly forecasts between sentences, that is, to use skip gram The basic unit is words instead of sentences. The specific way is to select a window, traverse the sentences, and then use the current sentence to predict and output its previous sentence and next sentence.

As for the sequence structure of RNN used in sentence modeling, when predicting the last and next sentences, it also uses the RNN of one sequence to generate every word in the sentence, so this structure is essentially an encoder decoder framework, but unlike the general framework, skip thoughts has two decoders.

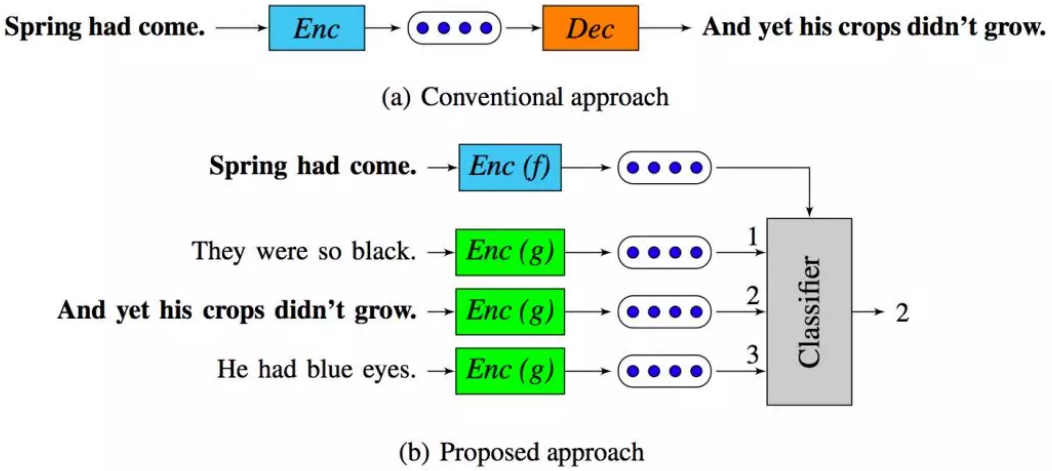
In today's view, there are many imperfections or improvements in this framework (the author also mentioned these future works separately in the paper). For example, the encoder of input can introduce the attention mechanism, so that the input of decoder no longer only depends on the output of the last moment of encoder; encoder and decoder can take advantage of deeper structure; decoder It can also continue to expand to predict more sentences in the context; RNN is not the only choice, such as CNN and the transformer structure proposed by Google in 2017 can also be used. Later, Google's Bert borrowed this idea, of course, this is the latter part, leaving the table.



10. Quick-thoughts

In 2018, on the basis of skip ideas, Google brain's logeswaran and others further improved this idea. They thought that the decoder efficiency of skip ideas was too low, and it could not be well trained on large-scale corpus (this is the common fault of RNN structure).

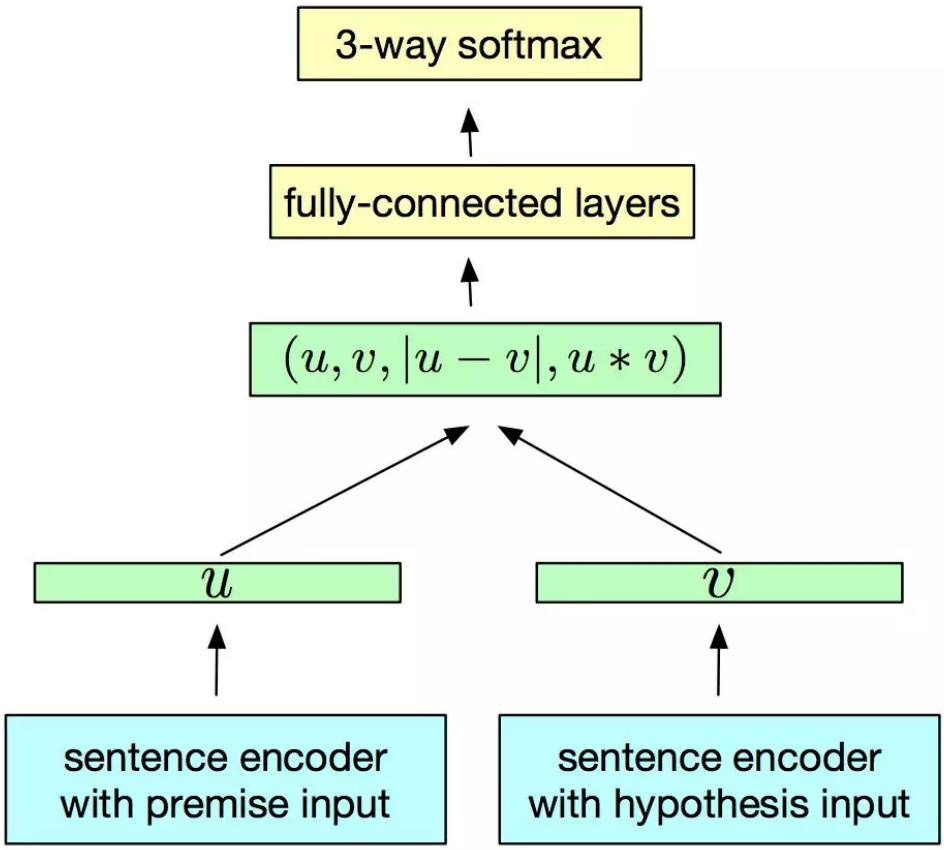
So they put skip ideas In particular, it is a classification task to mark sentence pairs in the same context window as positive examples and sentence pairs that do not appear in the same context window as negative examples, and to input these sentence pairs into the model so that the model can judge whether these sentence pairs are in the same context window. It can be said that just a few months later, Bert is taking advantage of this idea. These methods are all the same as skip thoughts.



11. InferSent

In addition to skip thoughts and quick thoughts, which do not need to mark data manually, there are also some ways to learn sentence representation from supervision data.

For example, in 2017, Facebook researcher conneau and others put forward the infersent framework. Its idea is very simple. First, design a model in SnLi (Stanford natural language) of Stanford The trained model is used as a feature extractor to obtain the vector representation of a sentence, and then the representation of the sentence is applied to the new classification task to evaluate the merits of the sentence vector. The frame structure is as follows:



The bottom layer of this framework is an encoder, that is, the final sentence vector extractor to be obtained. Then the obtained sentence vector is operated by some vectors to get the mixed semantic features of sentence pairs. Finally, the full connection layer is connected and the three classification tasks on SnLi are done. After doing the sentence matching task, we must know that this framework is the most basic (even the simplest) sentence matching framework. For the bottom layer of encoder, the author tried seven models respectively, and then used these models as the bottom layer of encoder structure to supervise and train on SnLi. After the training, the new classification task is evaluated, and it is found that when the encoder uses the bilstm with Max pooling structure, the sentence representation performance is the best

12. General Purpose Sentence Representation

In addition to infersent, a single task supervised learning, the latest work gradually

applies multi task joint learning to the representation of sentences.

For example, learning general purpose distributed sense representations via large scale multi task learning published on ICLR 2018 by Subramanian et al. Proposed four different monitoring tasks to jointly learn sentence representation, which are: natural language influence, skip ideas, natural machine translation and constitution Parsing et al.

The author's starting point is also very simple. General sentence representation should be learned by focusing on different tasks, rather than only one specific task. Later, the author's experiments in the paper also proved this point.

The specific method of the experiment is to train the above four tasks with the method of joint learning. After the training, the output of the model is used as the representation of the sentence (or the model of joint learning is used as the feature extractor), and then directly connect the representation with the non simple full connection layer as the classifier, and at the same time ensure the parameters in the lowest feature extractor Fixed (that is, only using it as a feature extractor), then training on the new classification task (only training the last full connection layer classifier), and finally evaluating on the test set of each classification task according to the trained simple classifier.

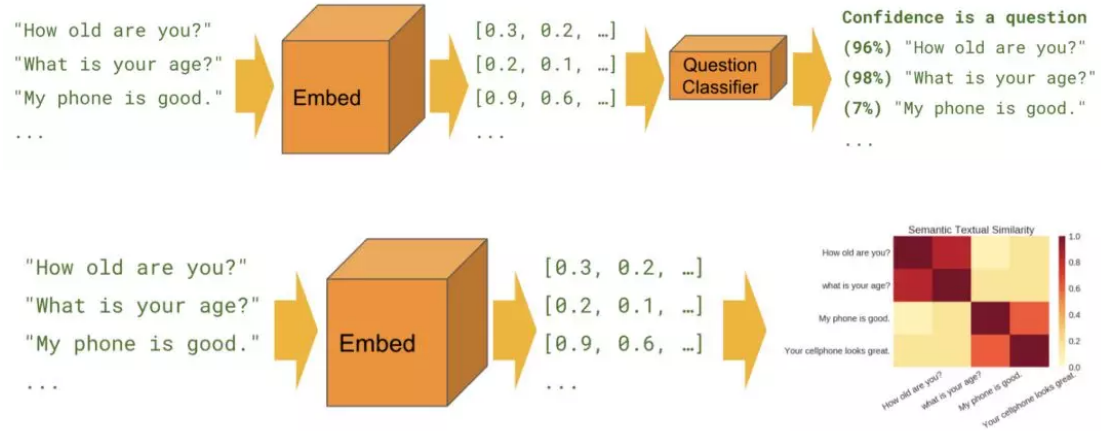
At last, the author found that their simple classifiers were better than the best results at that time in many tasks, and they also found that different tasks in joint training had different contributions to different aspects of sentence representation.

13. Universal Sentence Encoder

Similarly, in 2018, Daniel cer and others of Google put forward the same idea in the paper universal sense encoder as the work of general purpose sense representation, except that the author proposed using transformer and Dan (the deep unordered composition rivals synthetic methods for text mentioned above and similar to cbow and fasttext Classification) as the encoder of sentences.

Transformer structure is more complex, parameters are more, training is relatively time-consuming, but generally the effect will be better. Correspondingly, the Dan

structure is simple, with only two hidden layers (even reduced to only one hidden layer), less parameters, and relatively time-saving and resource-saving training. But generally speaking, the effect will be worse (not absolute, the paper also finds that the Dan effect is even better in some scenes). Then the author trains not only on unlabeled data, but also on supervised data. Finally, the author evaluates the transfer learning on ten classification tasks. In addition, the author also released their pre trained encoder, which can be used for sentence feature extractor of migration learning.

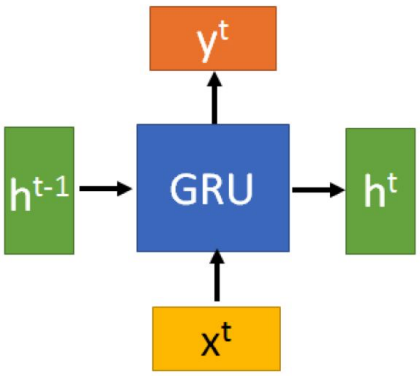


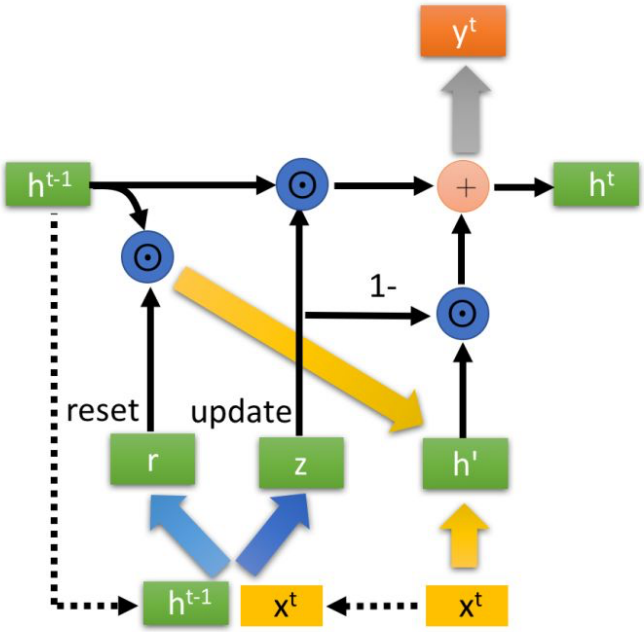
Pre-training language models

1. GRU

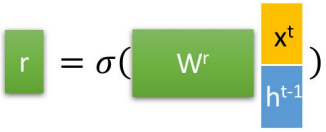
GRU (Gate Recurrent Unit) is a kind of RNN. Like LSTM (long short-term memory), it is also proposed to solve the problems of long-term memory and gradient in back propagation. GRU and LSTM are similar in many cases, but it' s more popular because GRU is computationally cheaper while remain similar performance.

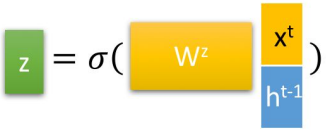
The structure of GRU:
Overall structure:



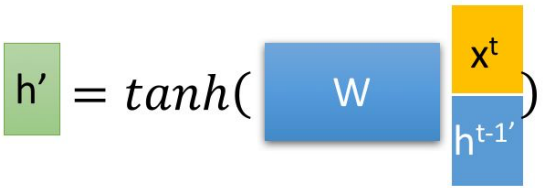


r: reset gate / z: update gate





After getting the gate control signal, first use reset gate control to get the data after "reset": $h^{t-1\prime} = h^{t-1} \odot r$. Then concatenate with xt and do tanh.



Here the h' mainly contains current input, xt. This process is similar with memorizing current state.

At last, we introduce the most critical step of GRU, which we can call "update memory" stage. At this stage, we have two steps at the same time: forgetting and remembering.

$$h^t = z \odot h^{t-1} + (1 - z) \odot h'$$

First of all, again, the range of gating signal (z) is from 0 to 1. The closer the gating signal is to 1, the more data it represents in memory, and the closer it is to 0, the less date it keeps.

What's smart about GRU is that we use the same gating to forget and select memories at the same time:

$z \odot h^{t-1}$: Indicates selective "forgetting" of the original hidden state. The formula z here can be thought of as a forget gate, forgetting some unimportant information in the formula dimension.

$(1 - z) \odot h'$: This formula indicates selective "memory" of h' which contains current node information. Similar to the above, the formula (1-z) here will also forget some unimportant information in the formula dimension. Or, we should consider it as a choice of some information in the formula dimension.

$h^t = z \odot h^{t-1} + (1 - z) \odot h'$: Combined with the above, the operation of this step is to forget some information in the h(t-1) passed down and add some dimension information entered in the current node.

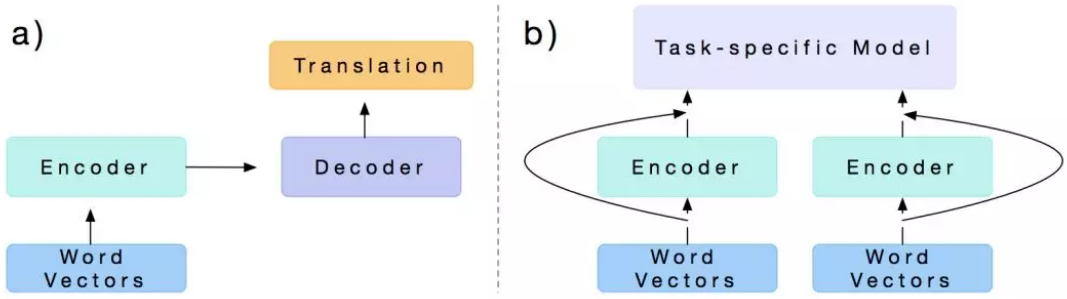
In summary:

The input and output structure of GRU is similar to that of RNN, and its internal idea is similar to that of LSTM.

Compared with the LSTM, there are less gates in the GRU, with fewer parameters than the LSTM, but it can also achieve the same function as the LSTM. Considering the computing power and time cost of the hardware, people usually prefer GRU.

2. CoVe

In 2017, Bryan McCann and others of salesforce published an article named learned in Translation: Contextualized word vectors. In this paper, we first used an encoder decoder framework to pre train the training corpus of machine translation, and then used the trained model to select only the embedding layer and encoder layer, and designed a task specific on a new task In the model, the output of the embedded layer and the encoder layer is used as the input of the task specific model, and finally the training is carried out in the new task scenario. They tried a lot of different tasks, including text classification, question answering, natural language influence and squad, etc., and compared the effects of these tasks with glove as the input of the model. The experimental results show that their proposed context vectors have brought different degrees of improvement in different tasks.



Different from the above mentioned skip thoughts methods, Cove focuses on how to migrate the pre trained representations from the existing data to the new task scenario, while most of the previous sentence level tasks only use the migration process as a means to evaluate their representation effect, so the concept is

different.

So, it seems that Cove has achieved an eye-catching result by pre training on surveillance data. Can we further pre train on unmarked data instead of relying on surveillance data?

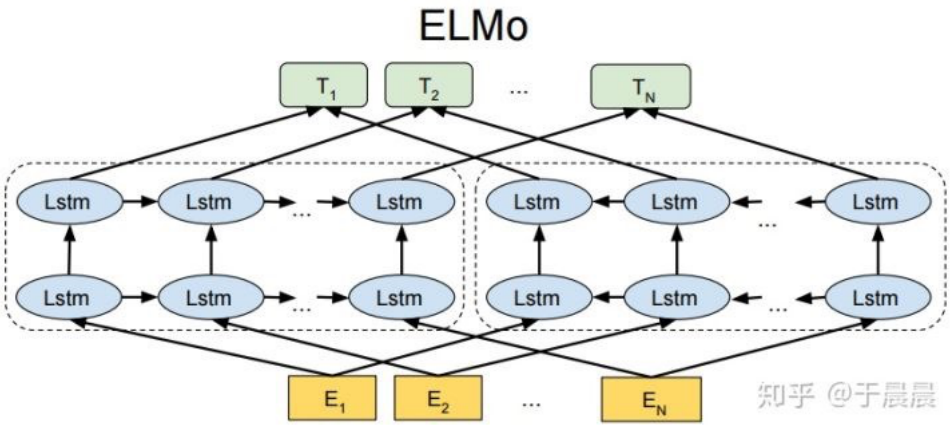
3. ELMo

The main reason why the effect of word2vec is not so amazing is that it is difficult to deal with polysemous words.

Elmo, its core idea is that the word vector changes at any time according to the context, that is, the dynamic word vector:

Different from the past, a word corresponds to a vector, which is fixed. In the Elmo world, the pre trained model is no longer just a vector correspondence, but a trained model. When using, input a sentence or a paragraph into the model, and the model will infer the word vector corresponding to each word according to the online text. After doing this, one of the obvious advantages is that for polysemous words, we can combine the context before and after to understand polysemous words. Apple, for example, can be interpreted as a company or a fruit according to the context of the previous and subsequent chapters

For LSTM, Elmo has two improvements. The first is to use multi-layer LSTM, and the second is to add backward LM.



The forward language model is:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, ..., t_{k-1})$$

The backward language model is:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, ..., t_N)$$

In this paper, the bidirectional LSTM language model is used, which combines the forward and backward language models and maximizes the joint log likelihood loss function of the forward and backward language models

$$\sum_{k=1}^{N} (logp(t_k | t_1, ..., t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}; \Theta_S) + logp(t_k | t_{k+1}, ..., t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}; \Theta_S))$$

For the model, there are three layers of word vectors that can be used: the output of CNN at the input layer, that is, the input vector of LSTM, the output vector of LSTM at the first layer and the output vector of LSTM at the second layer.

Because LSTM is bidirectional, for any word, if the number of layers of LSTM is l, the total number of vectors that can be obtained is 2L + 1, which is expressed as follows:

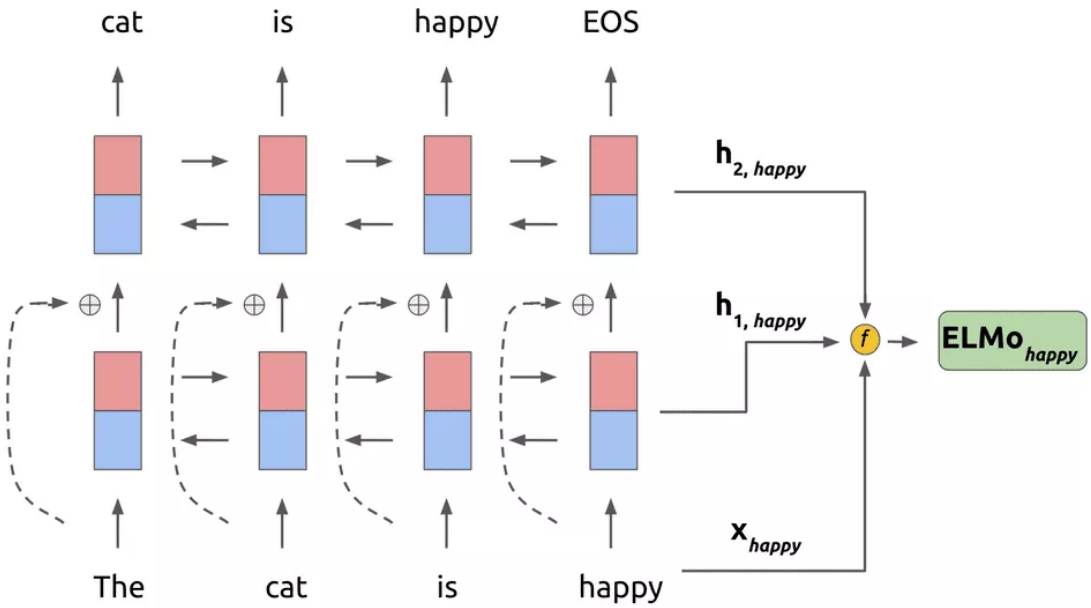
$$R_k = x_k, \overrightarrow{\mathbf{h}} k, j, \overleftarrow{\mathbf{h}} k, j, j = [1, 2, \ldots, L]$$

So far, we just extract the vector of Elmo. For each word, you can get its vector according to the following formula, where γ is a scale factor. The main purpose of adding this factor is to flatten the vector distribution of Elmo and the vector distribution of specific tasks to the same distribution level, which requires such a scale factor.

In addition, SJ uses a softmax parameter to learn the weight parameters of different layers according to the output vector of each layer. Because the meaning granularity of words required by different tasks is also different, it is generally believed that the representation of shallow layer tends to syntax, while the output

vector of high layer tends to semantic information. Therefore, it is a reasonable way to let tasks automatically learn the weight between each layer through a softmax structure.

$$\mathbf{ELMo}k^{task} = \gamma^{task} \sum j = 0^{L} s_j^{task} \mathbf{h}_{k,j}$$



Emlo is divided into two stages in practical application:

One is to use the language model for pre training. First, use the bilstm language model to train word embedding of words in large-scale corpus and record the vector expression of LSTM in each layer. After the pre training of bilstm, you can also perform fine in task corpus Tuning, which can reduce the confusion of language model and improve the effect of downstream tasks, can also be regarded as domain transfer;

Second, in the downstream supervised learning task, the task corpus is input into the input of bilstm. Through learning the weight parameters from the bottom to the top, different levels of vector expression are linearly combined, and it is added to the downstream task as a new feature. At this time, different contexts are adjusted by different combination parameters, and the effect of "dynamic adjustment" is achieved.

Advantage:

Deep, using bi LSTM to learn the complex characteristics of words, embodies the

syntactic and semantic features represented by different hidden layers of LSTM language model;
Contextualized, which combines different hidden layers and original embedding to learn polysemous words according to different contexts.

Disadvantages:
One obvious disadvantage is that Elmo uses LSTM instead of new and expensive transformer in feature extractor selection
The ability of Elmo to adopt two-way splicing is probably weaker than that of Bert integration. However, this is only a doubt generated from reasoning. At present, there is no specific experiment to explain this.

4. ERNIE

Google's Bert model, by randomly shielding 15% of words or words, makes use of transformer's multi-layer self-attention bi-directional modeling ability, and achieves good results in various NLP downstream tasks (such as sentence pair classification task, single sentence classification task, question answering task). However, the Bert model mainly focuses on the cloze learning of word or English word granularity. It does not make full use of the lexical structure, grammatical structure and semantic information in the training data to learn and model. For example, "I want to buy an Apple smartphone", the Bert model treats every word of "I", "want", "to", "buy", "an", "Apple", "smartphone" in a unified way, random mask, lost the information that "Apple phone" is a very popular noun, which is the lack of lexical information. At the same time, I + buy + noun is a very obvious sentence pattern of shopping intention. Bert does not specifically model this kind of grammar structure. If there is only "I want to buy an apple smartphone" and "I want to buy a Huawei smartphone" in the pre-training corpus, a new mobile phone brand such as chestnut mobile phone will appear one day, but this mobile phone brand does not exist in the pre-training corpus, Without the modeling based on lexical structure and syntactic structure, it is difficult to give a good vector representation for this new word. Ernie, through the unified modeling of

lexical structure, grammatical structure and semantic information in training data, greatly enhances the ability of general semantic representation, and has achieved the effect of greatly surpassing Bert in many tasks.

Here is the masking strategy difference:

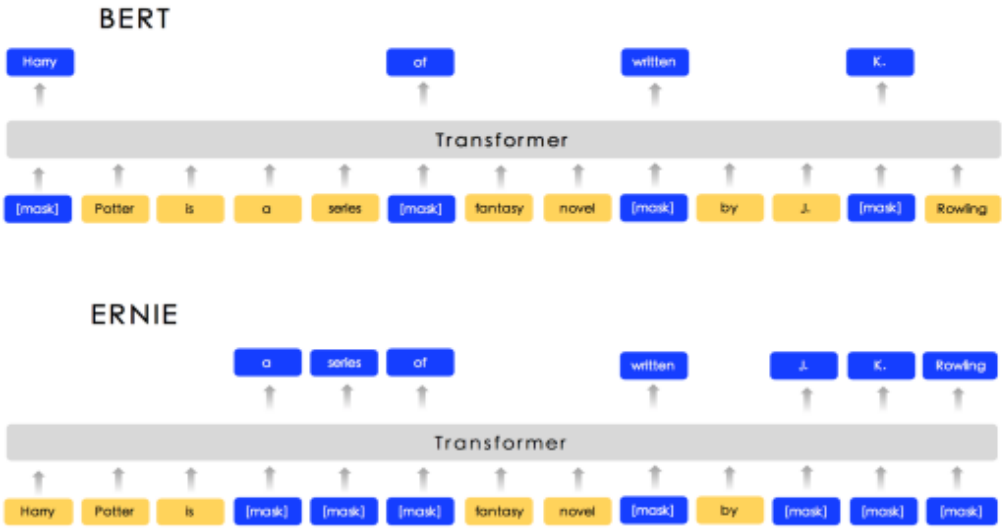
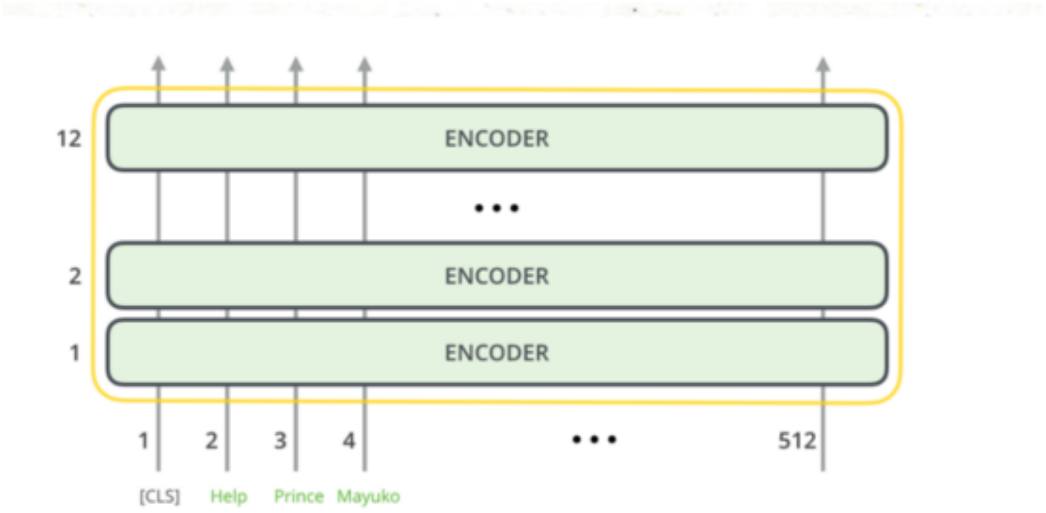


Figure 1: The different masking strategy between BERT and ERNIE

ERNIE Structure:



Compared with transformer, Ernie is basically the encoder part of transformer, and the encoders are all the same in structure, but do not share the weight. The specific differences are as follows:

Transformer: 6 encoder layers, 512 hidden units, 8 attention heads

ERNIE Base: 12 encoder layers, 768 hidden units, 12 attention heads

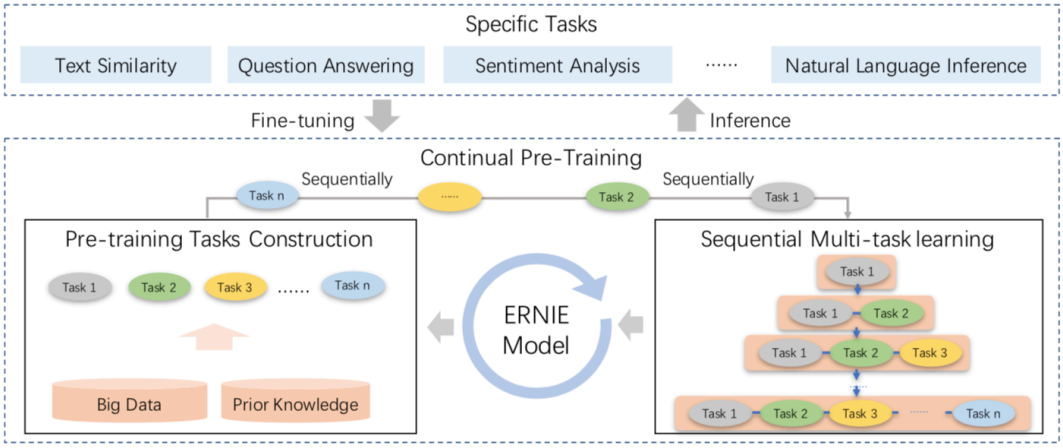
ERNIE Large: 24 encoder layers,1024 hidden units, 16 attention heads
From the input point of view, the first input is a special CLS. CLS means that the classification task is like the general encoder of transformer. Erine inputs a series of words into the encoder. Each layer uses self-attention, feed word network, and then passes the result to the next encoder.

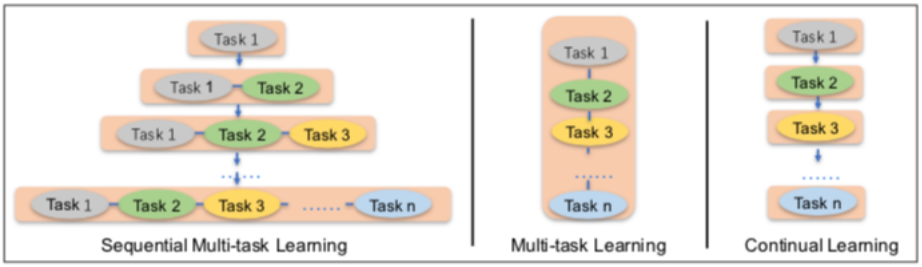
Compared with Bert, Ernie 1.0 improves two kinds of masking strategies, one is based on the phrase (like: a series of, written, etc.), the other is based on the entity (like: people's name, location, organization, product, etc., such as apple, J.K. Rowling). In Ernie, the phrase or entity composed of multiple words is regarded as a unified unit. Compared with Bert's word-based mask, all words in this unit are uniformly masked during training. Compared with mapping the query of knowledge class directly into vectors and then adding them up directly, Ernie can potentially learn the knowledge dependency and longer semantic dependency to make the model more generalized through the way of unified mask.

Sentence	Harry	Potter	is	a	series	of	fantasy	novels	written	by	British	author	J.	K.	Rowling
Basic-level Masking	[mask]	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	J.	[mask]	Rowling
Entity-level Masking	Harry	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]
Phrase-level Masking	Harry	Potter	is	[mask]	[mask]	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]

5. ERNIE2

The traditional pre training model is mainly based on the co-occurrence of words and senses in the text. In fact, it is also very important to train lexical structure, grammatical structure and semantic information in text data. In named entity recognition, nouns such as person name, organization name and organization name contain conceptual information corresponding to lexical structure, sentence order corresponding to grammatical structure, and semantic relevance corresponding to semantic information. In order to find this valuable information in training data, a pre training framework is proposed in Ernie 2.0, which can be used for incremental training in large data sets.





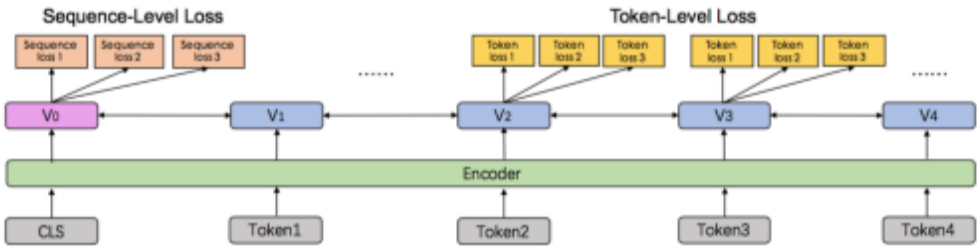
A very important concept in Ernie 2.0 is continuous learning. The purpose of continuous learning is to train several different tasks in a model in order to remember the results of the previous learning task in the next learning task. By using continuous learning, new knowledge can be accumulated continuously. The model can be initialized with the parameters learned from the history task in the new task. Generally speaking, it will get better effect than starting the new task directly.

part 1: Pre training continuous learning

Ernie's pre training continuous learning is divided into two steps. First, a large number of data and prior knowledge are continuously used to build different pre training tasks. Secondly, the Ernie model is constantly updated with pre training tasks.

For the first step, Ernie 2.0 constructs the pre training tasks of lexical level, grammatical level and semantic level respectively. All these tasks are based on dimensionless or weakly dimensioned data. It should be noted that before continuous training, a simple task is used to initialize the model first, and when updating the model later, the parameters trained by the previous task are used as

the initialization parameters of the next task model. In this way, whenever a new task is added, the parameter initialization of the previous model is used to ensure that the model will not forget the knowledge learned before. In this way, in the process of continuous learning, the Ernie 2.0 framework can constantly update and remember the previously learned knowledge, which can make the model perform better in new tasks. In the following e, F, G, we will specifically introduce what pre training tasks Ernie 2.0 builds and what role these pre training tasks play. In the figure below, the architecture of ernie2.0 continuous learning is introduced. This architecture includes a series of shared text encoding layers to encode context information. The parameters of encoder layers can be updated by all pre training tasks. There are two types of loss functions, one is the loss of sequence level, the other is the loss of word level. In Ernie 2.0 pre training, the loss functions of one or more sentence levels can be combined with the loss functions of multiple token levels to jointly update the model.



part 2: Encoder

Ernie 2.0 uses the transformer structure encoder. The structure is basically the same, but the weights are not shared.

part 3: task embedding.

Ernie 2.0 uses different task IDs to mark pre training tasks. Task IDs from 1 to n correspond to the pre training tasks below. The corresponding token segment position and task embedding are used as the input of the model.

pre-training task 1: Construct the pre training task of lexical level to obtain the lexical information in the training data

1: Knowledge masking task, that is, entity mask and phrase entity mask in Ernie 1.0, can acquire the prior knowledge of phrase and entity. Compared with sub word masking, this strategy can better capture the local and global semantic information of input samples.

2: Capital prediction task: capitalized words such as apple usually have specific meanings in sentences compared with other words, so add a task in Ernie 2.0 to determine whether a word is capitalized.

3: Token document relation prediction task, similar to TF IDF, predicts whether a word will appear in paragraph a and paragraph B of the text. If a word appears in many parts of an article, it means that it is often used or related to the topic of the article. By identifying the key words in the text, this task can enhance the ability of the model to obtain the key words of the text.

pre-training task 2: f: Build a grammar level pre training task to obtain the grammar information in the training data

1: Sense reordering task, during the training, divide the paragraph into 1-m segments randomly, and shuffle all the combinations randomly. Let's use the pre - trained model to identify the correct order of all these segments. This is a k-classification task:

$$k = \sum_{n=1}^{m} n!.$$

In general, these sense reordering tasks enable the pre trained model to learn

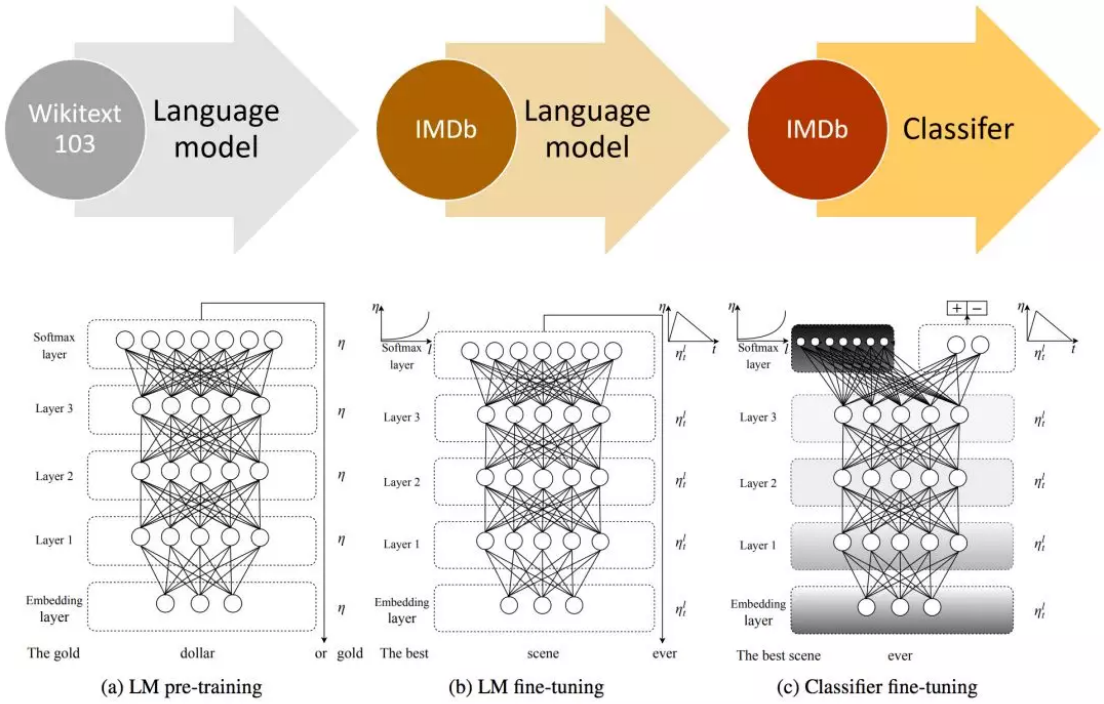
about the relationship between different senses in a document.
2: Sense distance task: construct a three-category task to judge the distance between sentences. 0 means two sentences are adjacent sentences in the same article, 1 means two sentences are in the same article, but not adjacent, and 2 means two sentences are different articles. By constructing such a three-classification task to judge the position relationship of sentence pairs (including three categories of adjacent sentences, non-adjacent sentences in documents, and sentences in non-same documents), better modeling semantic relevance.

pre-training task 3: Build a grammar level pre training task to obtain the grammar information in the training data
1: Sense reordering task, during the training, divide the paragraph into 1-m segments randomly, and shuffle all the combinations randomly. Let's use the pre - trained model to identify the correct order of all these segments. This is a k-classification task
In general, these sense reordering tasks enable the pre trained model to learn about the relationship between different senses in a document.
2: Sense distance task: construct a three-category task to judge the distance between sentences. 0 means two sentences are adjacent sentences in the same article, 1 means two sentences are in the same article, but not adjacent, and 2 means two sentences are different articles. By constructing such a three-classification task to judge the position relationship of sentence pairs (including three categories of adjacent sentences, non-adjacent sentences in documents, and sentences in non-same documents), better modeling semantic relevance.

6. ULMFit

The same thing with Elmo is that ulmfit also uses a language model, and the pre training model is mainly LSTM. The basic idea is to finish the pre training and finish it on specific tasks, but there are many differences.
First of all, the pre training and finetune process of ulmfit are mainly divided into

three stages. They are pre training on a large corpus (such as wikitext 103, with 103 million words), and then using the language model to finetune (the first finetune, called LM) on the specific task data Finetune), and then according to the specific task design model, take the pre trained model as the multi-layer of the task model, and finetune (the second finetune, if it is a classification problem, it can be called classifier finetune). The whole process is as follows:



Secondly, the model used comes from the paper regulating and optimizing LSTM language models of salesforce in 2017. In this article, they put forward awd-lstm. As the name reveals, this framework is more of a training method. The main idea is divided into two parts. Average SGD refers to training the model to a certain extent first Epoch, and then average the weight of each subsequent round to get the final weight, which is expressed by the formula, the common SGD method weight update process is as follows:

$$w_{k+1} = w_k - \gamma_k \nabla f(w_k)$$

Where k represents the number of iterations, and F is the loss function, which is a common SGD weight update iteration. Then ASGD turns it into:

$$w = \frac{1}{K - T + 1} \sum_{i=T}^{K} w_i$$

Where t is a threshold value, and K is the total number of iterations. This formula means that after iterating to the t-th time, all values of the parameter between the t-th turn to the last turn are averaged to get the parameter value of the final model. Accordingly, the general SGD takes w = wk as the parameter value of the final model directly.

In addition to using ASGD method to train the model, there is a connection between the hidden layers at one time and the next time in the general LSTM, and the connection is connected through a fully connected matrix, while the model uses dropconnect method to randomly drop Some connections are dropped, which reduces the risk of over fitting. Of course, there are normal dropout operations between the input layer and the hidden layer.

Third, the design of fine-tuning method is very exquisite. The author puts forward several fine-tuning techniques, which are: discriminative fine tuning, sloped triangular learning rates and gradual unfreezing.

The basic idea of discriminative fine tune is to give different learning rates to different layers when updating parameters. The starting point here is that for the NLP deep learning model, the representations of different layers have different physical meanings, such as shallow partial syntactic information, high partial semantic information, so for different learning rates, it is naturally more reasonable. The specific formula is as follows:

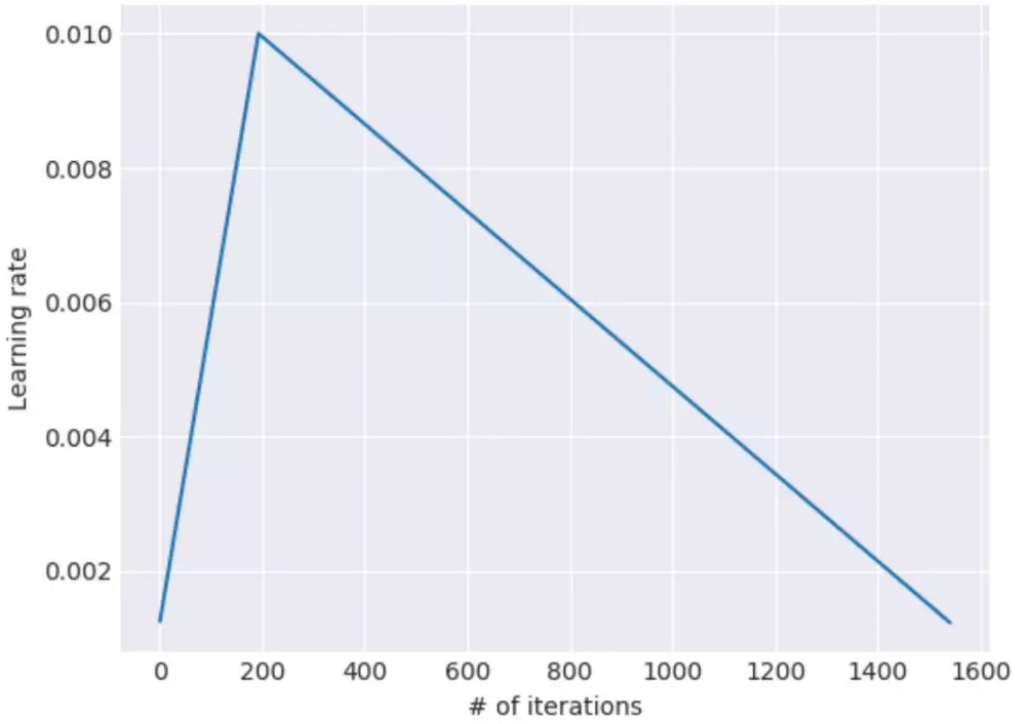
$$\theta_t^l = \theta_{t-1}^l + \eta^l \nabla_{\theta^l} J(\theta)$$

Here η^l is the different learning rates of different layers. The original text also gives specific choices: first, specify the learning rate of the last layer, and then get the learning rate of the front layer according to the following formula. The basic idea is to make the learning rate of the shallow layer smaller.

$$\eta^{l-1} = \frac{\eta^l}{2.6}$$

For the slotted triangular learning rates, the main idea is to stabilize the parameters that have been pre trained on the large-scale corpus in the first stage of finetune, so choose a relatively small finetune learning rate. Then hope to gradually increase the learning rate, so that the learning process can be as fast as possible. When the training is close to the end, the learning rate is gradually reduced, so that the model gradually converges smoothly.

This idea, I think, is based on the learning rate adjustment method of warm up proposed by Google in 2017, which is also used to gradually increase the learning rate and then gradually reduce it during training. Therefore, such a syllogism learning process is illustrated as follows:



Another finetune technique is gradual unfreezing. The main idea is to unfreeze the model layer by layer when the pre training model is applied to finetune on a new task. That is to say, the last layer of finetune is first unfrozen, and then the last layer is unfrozen. The last layer and the last layer are together to finetune, and then the third layer is unfrozen. In this way, we can push forward layer by layer to

the shallow layer, and finally the whole model of finetune will terminate to a certain middle layer. The goal is to make the finetune process more stable.

Of course, it is worth mentioning, because ulmfit includes two finetunes, that is, using language model finetune on new tasks and training a final task specific model (such as classifier) on new tasks.

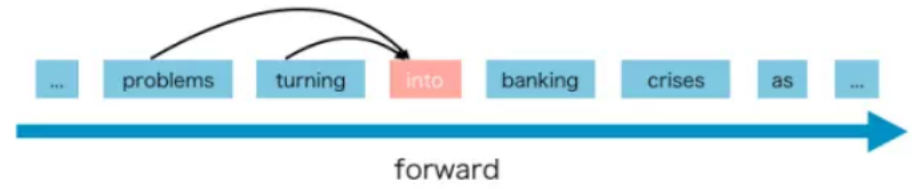
In this paper, two techniques of discriminative fine tuning and sloped triangular learning rates are mainly used in the finetune stage of the language model, and the last grand unfreezing technique is applied in the finetune stage of the final task specific model.

Through the above methods, ulmfit finally performs amazing in the classification task, especially it only needs 100 tag data to learn a very comparable classifier. It has to be said that the pre-training language model plays a crucial role in the final performance.

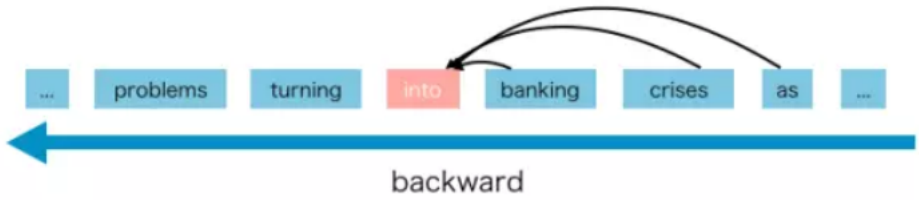
7. XLNet

Autoregressive (AR) language model

Ar language model is a model that uses context words to predict the next word. But here, context words are limited to two directions, forward or backward. Before Elmo / Bert came out, the commonly used language model was actually to predict the next possible following word according to the above content, that is, the language model task from left to right, or vice versa, that is, to predict the previous word according to the following. This type of LM is called autoregressive language model.



前向



后向

GPT and gpt-2 are both ar language models. Although Elmo seems to make use of the above and the following, it is still autoregressive LM in essence, which has something to do with how to implement the model. Elmo has made two directions (left to right and right to left language models), but it has two directions of autoregressive LM respectively, and then splices the hidden node states of the two directions of LSTM together to reflect the two-way language model. So it is actually a combination of two autoregressive language models, which is still an autoregressive language model in essence.

The advantage of AR language model is that it is good at generating natural language processing tasks. Because when generating context, it is usually forward-looking. Ar language model is naturally suitable for such NLP tasks.

However, AR language model has some disadvantages. It can only use forward context or backward context, which means it can not use both forward and backward context. Of course, it seems that Elmo can do both ways, and then splicing seems to solve this problem. Because the fusion mode is too simple, the effect is not very good.

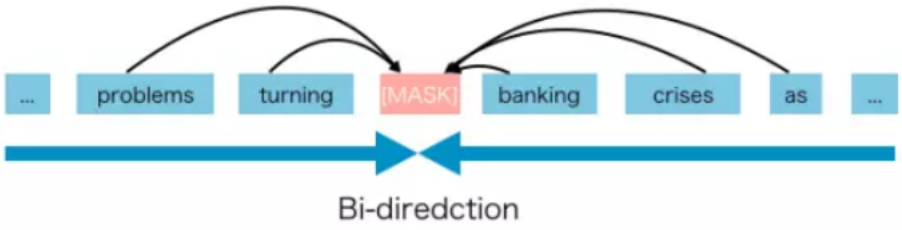
Automatic encoder (AE) language model

AE language model aims to reconstruct the original data from the damaged input, randomly mask some words in input x, and then one of the main tasks of the pre

training process is to predict these words dropped by mask according to the context words.

Corrupted input means that we replace the original word into with [mask] in the pre training phase. The goal is to predict into to get the original sentence.

The advantage of the AE language model is that it can naturally integrate into the two-way language model, and see the above and below of the predicted words at the same time.



双向

However, AE language model has its disadvantages. It uses [mask] in pre training, but this artificial symbol does not exist in the real data when tuning, which will lead to the difference between pre training and tuning. Another disadvantage of [mask] is that it assumes that prediction (masked) words are independent of each other given unshielded words. For example, we have a saying that "it shows that the housing crisis has become a banking crisis". We mask "banking" and "crisis". Note here that we know that the masked "banking" and "crisis" contain an implicit relationship with each other. However, AE model tries to predict that "banking industry" will give unshielded words and "crisis" will give unshielded words respectively. It ignores the relationship between "banking" and "crisis". In other words, it assumes that the predicted (masked) markers are independent of each other. But we know that the model should learn to predict the correlation between (masking) words to predict one of them.

What the author wants to emphasize is that xlnet proposes a new method to let ar language model learn from two-way context, so as to avoid the disadvantages of mask method in AE language model. It still looks like a left to right input and prediction mode, but in fact, the following information of the current word has been introduced internally? Xlnet's main contribution to the model is actually here.

Model method

Xlnet still follows a two-stage process, the first stage is the language model pre training stage, and the second stage is the task data fine tuning stage. It mainly wants to change the first stage, that is, instead of Bert's denoising autoencoder mode with mask symbols, it adopts the autoregressive LM mode. That is to say, it seems that input sentence x is still input from left to right. See the above context "before" of Ti word to predict the word ti. But I also hope that in context before, I can see not only the above words, but also the following words in context after. In this way, the mask symbols introduced in the pre training phase of Bert are not needed. So in the pre training phase, it seems to be a standard left to right process, and fine tuning is also the process, so the two links are unified Come along.

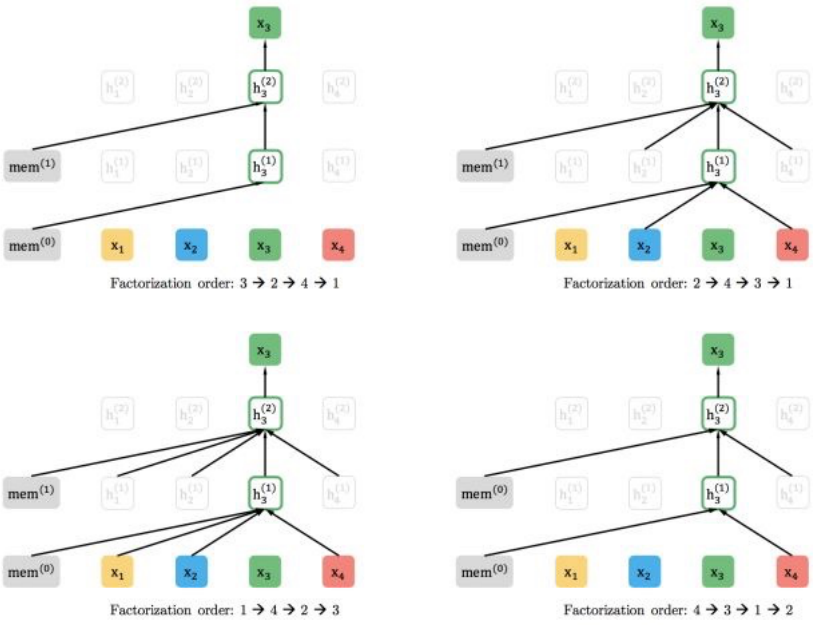


Figure 1: Illustration of the permutation language modeling objective for predicting x_3 given the same input sequence $\mathbf{x}$ but with different factorization orders.

Xlnet introduces the training objective of permutation language model in the pre training stage. What do you mean? That is to say, for example, the current input sentence x containing the word ti is composed of several words in order, such as x1, X2, X3, X4. Let's assume that the word Ti to be predicted is X3, and the position is position 3. To make it be able to see the word X4 of position 4 in the context "before" above, that is, position 1 or position 2. Let's do this: suppose we fix the position of X3, that is, it is still in position 3, and then randomly arrange and

combine the four words in the sentence. In the various possibilities after randomly arranging and combining, select another part as the input X of model pre training. For example, after random permutation and combination, x4, X2, X3 and X1 are extracted as the input X of the model. So, X3 can see x2 above and X4 below at the same time. This is the basic idea of xlnet. So, after reading this, we can understand its original intention: it still looks like an autoregressive left to right language model, but in fact, by arranging and combining the words in the sentence, a part of the words below Ti are placed in the above position of Ti, so we can see the above and the following, but the form is still From left to right, predict the next word.

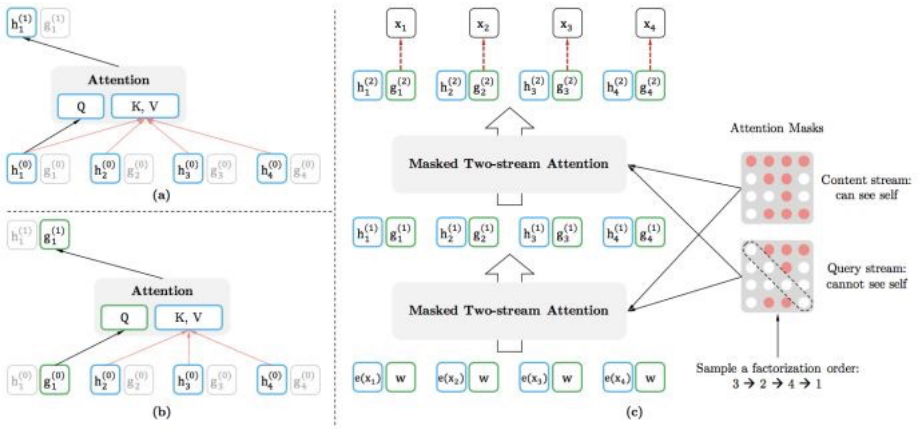


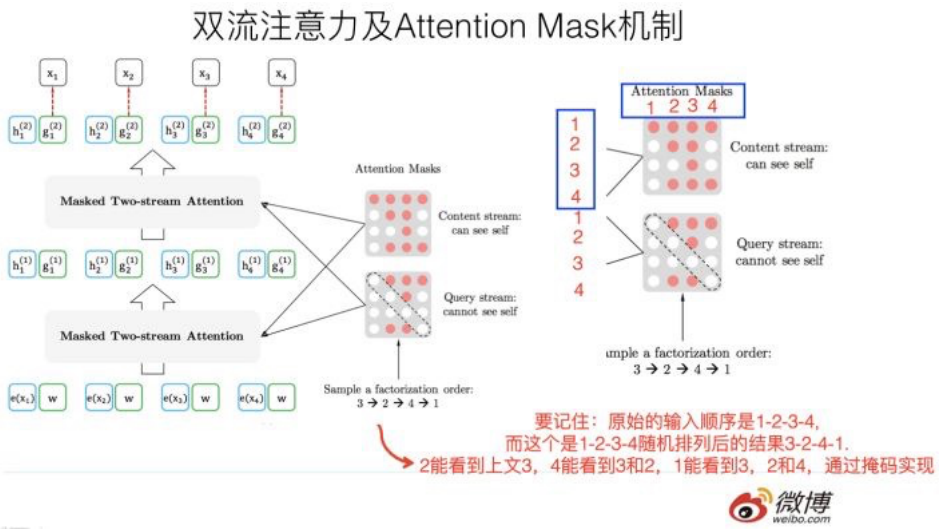
Figure 2: (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content x_{z_t}. (c): Overview of the permutation language modeling training with two-stream attention.

First of all, we need to emphasize that although the above is to arrange and combine the words of sentence x, then randomly select examples as input, in fact, you can't do this, because you can't arrange and combine the original input in the final tuning stage. So, we must make the input part of the pre training stage still look like the input sequence of x1, X2, X3, x4, but we can do some work in the transformer part to achieve our desired goal. Specifically, xlnet adopts the mechanism of attention mask. You can understand that the current input sentence is x, the word Ti to be predicted is the ith word, and the first 1 to I-1 words do not change in the input part. Who or who should be. But in transformer, through the attention mask, I-1 words are randomly selected from the input words of X, i.e. the words above and below Ti, and put them in the positions above Ti, and hide the input of other words through the attention mask, so that we can achieve our

desired goal (of course, the so-called place above ti is just an image statement, In fact, in the interior, through attention mask, other words that have not been selected are masked out to prevent them from working when predicting the word ti. That's all. It's similar to putting the selected words in the position of context "before". When implementing, xlnet is implemented with "double flow self attention model".

Double flow self attention mechanism

Query flow self attention: what is this? In fact, it is used to replace the [mask] mark of Bert, because xlnet wants to discard the [mask] mark, but for example, it knows the word X1 above, X2, to predict the word X3, you can predict the word X3 at the highest level of the transformer corresponding to the position X3, but you can't see the word X3 to be predicted on the input side. Bert actually directly introduces the [mask] tag to cover the content of the word X3, which means that [mask] is a general placeholder. Because xlnet wants to discard the [mask] tag, but it can't see the input of X3, the query stream directly ignores the input of X3, and only keeps the location information, and uses the parameter w to represent the embedding code of the location. In fact, xlnet just discards the [mask] placeholder on the surface, but introduces query flow to ignore the word masked. Compared with Bert, it's just a different way of implementation.



Content flow from attention: in fact, it is the standard transformer computing process.

Attention The core of mask's mechanism is that although the current input still looks like X1 - > X2 - > X3 - > x4, we have changed it to another order of random permutation and combination, X3 - > X2 - > X4 - > x1. If this example is used to train LM from left to right, it means that when X2 is predicted, it can only see X3 above; when X4 is predicted, it can only see X3 and X2 above, and so on So, for example, for X2, we can see the following X3. This maintains the surface word order of X sentences on the input side, but in fact, what we see inside transformer is the order after rearrangement and combination, which is realized by attention mask. As shown in the above figure, the input still looks like x1, X2, X3, X4. You can use different mask matrices to make the current word Xi only see the words in front of you in the order of X3 - > X2 - > X4 - > X1 after being arranged and combined. This internally changes to the predicted word and sees the context word, but the input side still seems to maintain the original word order. The key is to see the mask matrix on the right side of the figure above. I believe that many people didn't see it at the beginning, because I didn't see it at the beginning, because the word coordinates of the mask matrix are not marked. Its coordinates are 1-2-3-4, which is the word order of the x on the surface. Through the mask matrix, you can change the permutation and combination you want, and let the current word see what it should see Article, in fact, is mixed with the above and the following content. This is what attention mask means to realize permutation and combination.

The similarities and differences with the pre training process of Bert

Although it seems that the new pre training goal of permutation language model introduced by xlnet in the pre training mechanism is quite different from the way Bert uses mask marking. In fact, if you think deeply, you will find that the two are similar in nature. The main difference is that Bert directly hides some words on the input side by introducing the mask tag, so that these words do not play a role in the prediction, and requires using other words in the context to predict a word that is dropped by the mask; xlnet discards the mask tag on the input side, and uses attention In the mask mechanism, a part of the words are randomly dropped in the transformer (the proportion of the words dropped by the mask is related

to the position of the current word in the sentence. The higher the position is, the higher the proportion of the words dropped by the mask is, the lower the proportion of the words dropped by the mask is), so that these words dropped by the mask do not work when predicting a word. So, in essence, there's not much difference between the two. It's just the location of the mask. Bert is more superficial. Xlnet hides this process in the transformer. In this way, we can discard the [mask] mark on the surface and solve the problem of inconsistency between the pre training with [mask] mark and the final tuning process. As for xlnet, the mutual independence of masked words in Bert, that is to say, when predicting a masked word, other masked words don't work. If you think about it deeply, it doesn't matter, because xlnet is in internal attention When a mask is used, a certain proportion of context words will be dropped by the mask. As long as there are some words dropped by the mask, in fact, they will face this problem. However, if the training data is large enough, it can make up for the direct interrelationship of the masked words by other examples instead of the current one, because there are always other examples that can learn the interdependence of these words.

I believe that the pre training process of Bert can simulate the process of permutation language model of xlnet. The current approach of Bert is to randomly mask 15% of the words given the input sentence x, and then ask to use the remaining 85% of the words to predict any word dropped by mask. The words dropped by mask do not play a role in this process. If we transform Bert's pre training process into: for the input sentence, randomly select any word Ti, and only change this word to mask mark. Suppose ti is the ith word in the sentence, then randomly select any I word in X, and only use this I word to predict the word dropped by mask. Of course, in theory, this process can also be implemented in transformer by using the attention mask. If so, Bert's pre training mode is basically equivalent to xlnet.

Or think about it from another perspective. Suppose we still use Bert's current mask mechanism, but extreme the condition that 15% of the mask is dropped. Instead, we can only drop one word per sentence, and use the remaining words to predict the words dropped by the mask. In fact, this process is similar to the PLM of xlnet. The main difference is that when predicting the words dropped by the mask, the context used is more. (when xlnet is implemented, in order to

improve efficiency, it actually selects the last 1 / K word of each sentence to be predicted. Assuming k = 7, it means that a sentence x, only the last 1 / 7 of the words will be predicted What does that mean? It means that at least 6 / 7 of the context words are reserved to predict a certain word. For the last word, it means that other words of X in all sentences are reserved, which is the same as the above mentioned Bert only keeps one masked word. Or we can consider xlnet from the perspective of Bert's pre training. If xlnet is changed to sentence x, only the last word in the sentence needs to be predicted, instead of the last 1 / K (assuming that K is particularly large), then in fact, only one word will be dropped from each input sentence of Bert, and the two are basically equivalent.

Of course, the xlnet transformation maintains the left to right model of the seemingly autoregressive language model, which Bert can't do. The obvious advantage is that for the task of generating classes, under the premise of maintaining the left to right generation process of the surface, the context information is implied in the model. So it seems that xlnet should have an obvious advantage over Bert in generating NLP tasks of type. In addition, because xlnet also introduces the mechanism of transformer XL, it will also have obvious advantages over Bert for NLP tasks of long document input type.

To sum up:

1. A new pre training objective different from Bert's de noising autoencoder: permutation language model (PLM for short), which can be understood as how to take specific measures to integrate the two-way language model in the autoregressive LM mode. This is xlnet's great contribution to the model, and it really opens a new way of thinking about the two-stage model trend in NLP.
2. The main idea of transformer XL is introduced: relative position coding and segmented RNN mechanism. Practice has proved that these two points are very helpful for long document task;
3. Increase the data scale used in the pre training stage; the pre training data used by Bert is books corpus and English wiki data, with a size of 13g. In addition to using these data, xlnet also introduces giga5, clueweb and common crawl data, and discards some of the low-quality data, which are 16g, 19g and 78g respectively. It can be seen that the data scale is greatly expanded in the pre

training stage, and the quality is filtered. The obvious route is gpt2.0.

8. Transformer

Preface

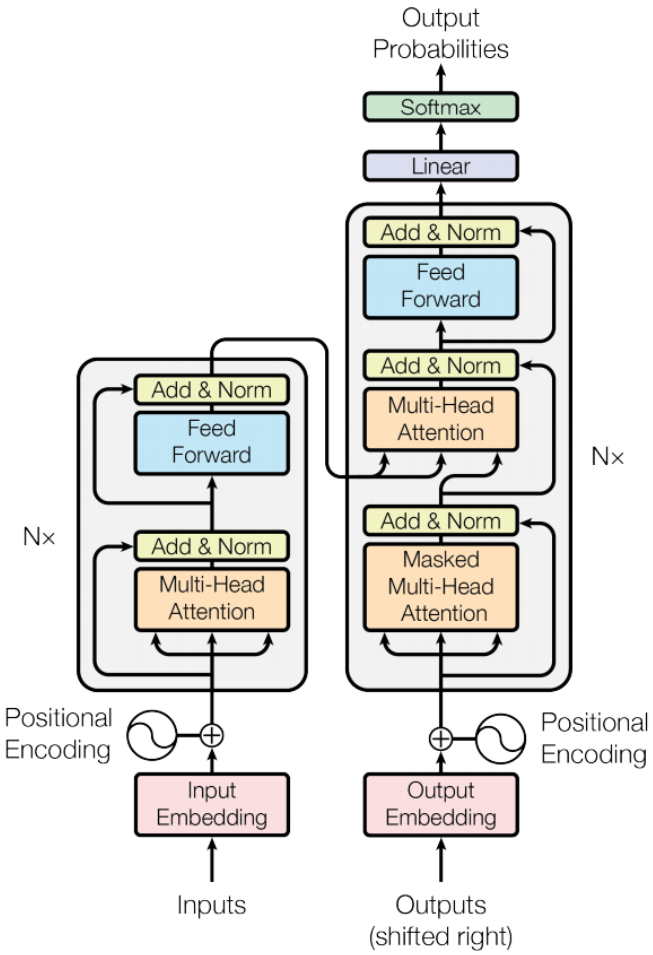
In June 2017, Google published a paper "attention is all you need" and proposed a transformer model. As the name of the paper says, it aims to replace the RNN cycle mechanism with attention mode, so as to parallel computing and speed up. At the same time, in a specific task, this model also surpassed the Google neural machine translation model at that time. The author mainly read the paper and two blogs (see the portal at the end of the article for the links), here is mainly to integrate and refine these contents~

I. background

Before the advent of transformer, RNN series networks such as LSTM, Gru and encoder decoder + attention architecture basically created the iron barrel of all NLP tasks. However, one of the defects of RNN is that it is an autoregressive model, which can only be calculated step by step and serially, and can not be parallelized. Therefore, some networks, such as bytenet and convs2s, use CNN as the basic building block to calculate the hidden layer representation of all input and output positions in parallel. However, in these models, the number of operations needed to correlate signals from two arbitrary input or output locations will increase with the distance between the locations, such as the linear growth of convs2s and the logarithmic growth of bytenet, which makes it more difficult to learn the dependence between distant locations. In transformer, the distance between two inputs has no effect on its calculation, and it is the same. It does not use RNN and convolution, so it can carry out parallel calculation.

2、 Overall framework of transformer

The following is the overall structure of transformer cut out from the paper:



3、 Analysis of transformer details

1. encoding

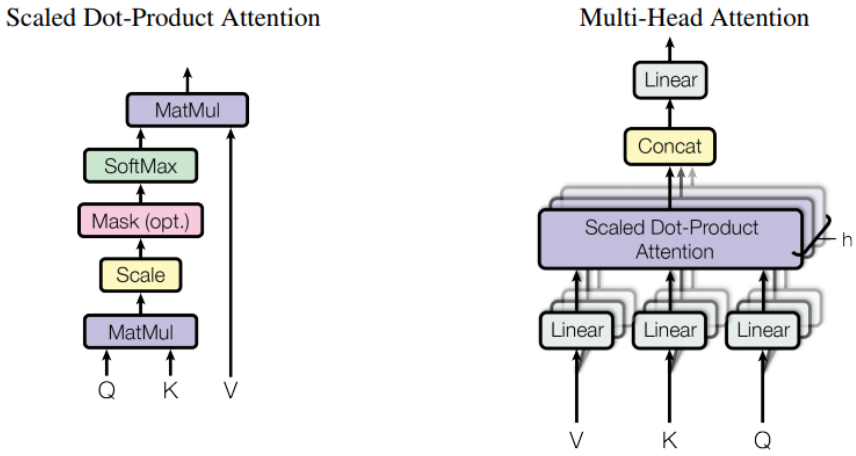
First, let's look at the encoder part (left half), which is stacked by the contents in the n-layer box. For each layer, it is composed of two parts: one is multi head self attention mechanism, the other is a simple fully connected feedforward network. In each part, residual + layer normalization is used for processing. In this paper, there are six such boxes, i.e. n = 6, and the number of hidden layer units of the model dmodel = 512

2. Self attention mechanism

Instead of RNN, encoder uses a self attention mechanism.

In general, the attention mechanism we use can be abstracted as inputting a query to query the key in the key value pair, and then we get a probability distribution, and then we add the values to obtain the attention representation under the current query. Our query is often the output of a step in the decoder, and the key value pair is often the output of the encoder.

This attention mechanism is also used in the paper, except that its query, key and value are all generated by the output of encoder through different transformations, that is, self attention. All things are themselves. They defined a calculation method called "scaled dot product attention", which is used to calculate the attention representation under given query, key and value, as shown in the following figure (left)



The calculation formula is

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Generally, we often use two methods of attention calculation: one is multiplicative attention, that is, using inner product; the other is additive attention, that is, using an additional layer of hidden layer to calculate. In theory, the complexity of the two methods is similar, but the multiplicative attention can save more time and space because it can use matrix operation. Compared with the above (left) and formula, the only difference between this formula and multiplicative attention is that a scaling factor is used:

$$\frac{1}{\sqrt{d_k}}$$

Why zoom here? The explanation is given in the paper: when the DK is small, the effect of no scaling is similar to that of the additive attention, but when the DK is large, the effect of no scaling is obviously worse than that of the additive attention. It is suspected that when the DK is growing, the magnitude of the inner product

will also increase, resulting in the softmax function being pushed to the region with smaller gradient. In order to alleviate this problem , plus this scaling item to reduce the magnitude.

It is also mentioned in the paper that the calculation method of using only one attention is too thin, so they propose a multi head attention mechanism. The calculation of attention is divided into different subspaces in order to learn attention from many aspects, as shown in the above figure (right). In parallel, Q, K and V are mapped to different spaces through different mapping matrices (each space is a head), and then a single "scaled dot product attention" is learned in these spaces respectively. Finally, the multi head attention representation is spliced and mapped to the original space through an additional mapping layer. The formula is as follows:

$$\begin{aligned}\mathrm{MultiHead}(Q,K,V) &= \mathrm{Concat}(\mathrm{head}_1,...,\mathrm{head}_\mathrm{h})W^O \\ \text{where } \mathrm{head}_\mathrm{i} &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Decoder

Next, let's look at the decoder part (the right half). It is also stacked by N layers (in this paper, n = 6). For each layer, in addition to the same self attention and feed as in encoder In addition to forward, a layer of attention layer in the traditional encoder decoder framework is inserted in the middle, that is, the output of the decoder is used as query to query the output of the encoder, and multi head attention is also used, so that all the output of the encoder can be seen during the decode.

At the same time, as a decoder, when predicting the current step, we can't know the later content, that is, attention needs to add mask, set all the scores after the current step to $-\infty$, and then calculate softmax to prevent data leakage.

Location coding layer

Careful readers may find that there is another thing called positional encoding in the overall architecture diagram. What is it?

Although transformer discards the cyclic structure of RNN and the local

correlation of CNN, the most important thing for sequence is actually sequence. Looking at the previous self attention processing method, it is actually no different from the "word bag" model, so the defect of ignoring the location information must be made up by certain means.

In this paper, a very "smart" way is proposed to add location information, that is, positional encoding, which encodes each location pospospos, and then adds it with the word embedding of the corresponding location to form a new word embedding of the current location. It uses the following formula to code each pos:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Where III represents the position in the embedding vector, i.e. each dimension in the dmodel. There are two advantages to choose this sin function: 1) it can be coded directly without training, and the coding result can be obtained directly no matter what length; 2) it can represent the relative position according to the:

$$sin(\alpha+\beta) = sin\alpha cos\beta + cos\alpha sin\beta, \;\; PE_{pos+k}$$

It can be expressed as a linear transformation of pepos, which provides the possibility of expressing relative position information.

9. Transformer-XL

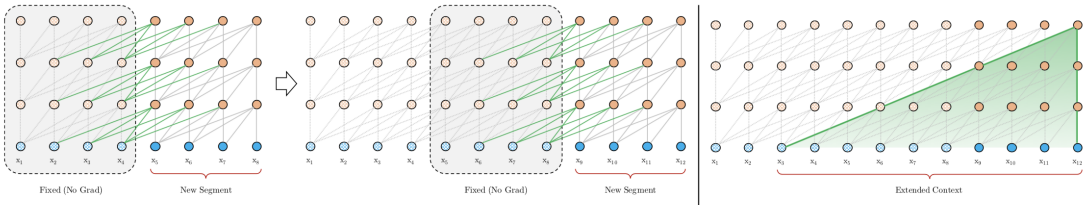
RNN learns the relationship between the input words or characters one by one according to the sequence order, while transformer receives a whole sequence, and then uses self attention mechanism to learn the dependency between them. Both of these architectures have made remarkable achievements at present, but they are limited to capturing long-term dependence.

Based on the vanilla transformer, the transformer XL architecture introduces two innovations: recurrence mechanism and relative positional encoding to overcome the shortcomings of vanilla transformer. Another advantage of transformer-xl

over vanilla transformer is that it can be used for word level and character level language modeling.

1. Introduce cycle mechanism

The essential difference between vanilla transformer and vanilla transformer is that it introduces a cycle mechanism between segments, so that the current segment can use the information of the previous segment to achieve long-term dependency when modeling



(a) Training phase. (b) Evaluation phase.

In the training phase, each hidden layer receives two inputs when processing the following segments:

1. The output of the hidden layer in front of the segment is the same as that of the vanilla transformer (gray line in the above figure).

2. The output of the hidden layer in the front section (the green line in the figure above) can make the model create a long-term dependency.

The two inputs are spliced and used to calculate the key and value matrices of the current segment. The specific calculation formula for a certain layer of a segment is as follows:

$$
\begin{aligned}
\widetilde{\mathbf{h}}_{\tau+1}^{n-1} &= [\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}], && \text{(extended context)} \\
\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} &= \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top}, && \text{(query, key, value vectors)} \\
\mathbf{h}_{\tau+1}^{n} &= \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right). && \text{(self-attention + feed-forward)}
\end{aligned}
$$

In the test phase, it will also be faster than vanilla Transformer. In the vanilla transformer, only one step can be advanced at a time, and the segments need to be rebuilt and calculated from scratch; in the transformer XL, the whole segment can be advanced at a time, and the data of the previous segment can be used to predict the output of the current segment.

2. Relative position coding

In transformer, an important thing is that it considers the location information of

the sequence. In the case of segmentation, if only the position coding in transformer is used directly for each segment, that is, the representation of each different segment in the same position uses the same position coding, there will be problems.

In transformer-xl, the above-mentioned calculation method of attention is transformed into the calculation of relative position, not only in the first layer, but also in each layer.

$$\mathbf{A}_{i,j}^{\text{rel}} = \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(b)} + \underbrace{u^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(c)} + \underbrace{v^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(d)}.$$

To interpret this formula from another perspective, you can divide the calculation of attention into the following four parts:

a. Content based "addressing", that is, the original score without adding the original location encoding.

b. Content based position offset, that is, the position deviation relative to the current content.

c. Global content bias, used to measure the importance of key.

d. The global position offset adjusts the importance according to the distance between query and key.

3. Overall calculation formula

Combined with the above two innovations, the overall calculation formula of transformer XL model is sorted out as follows. Here, we consider a model with only one attention head in N layer:

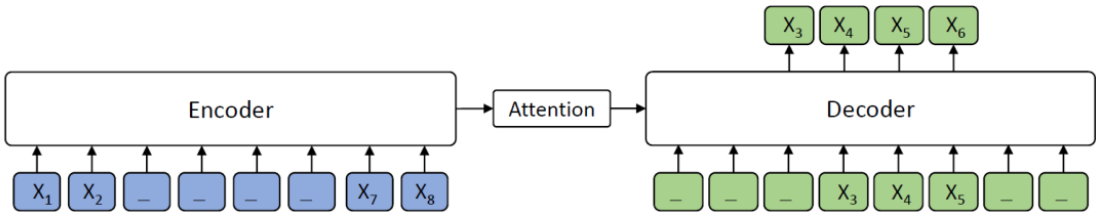
$$\begin{aligned}
\text{For } n = 1, \ldots, N: \qquad \widetilde{\mathbf{h}}_{\tau}^{n-1} &= \left[\text{SG}(\mathbf{m}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau}^{n-1}\right] \\
\mathbf{q}_{\tau}^{n}, \mathbf{k}_{\tau}^{n}, \mathbf{v}_{\tau}^{n} &= \mathbf{h}_{\tau}^{n-1}\mathbf{W}_q^{n\top}, \widetilde{\mathbf{h}}_{\tau}^{n-1}\mathbf{W}_{k,E}^{n}{}^{\top}, \widetilde{\mathbf{h}}_{\tau}^{n-1}\mathbf{W}_v^{n\top} \\
\mathbf{A}_{\tau,i,j}^{n} &= \mathbf{q}_{\tau,i}^{n}{}^{\top}\mathbf{k}_{\tau,j}^{n} + \mathbf{q}_{\tau,i}^{n}{}^{\top}\mathbf{W}_{k,R}^{n}\mathbf{R}_{i-j} + u^{\top}\mathbf{k}_{\tau,j} + v^{\top}\mathbf{W}_{k,R}^{n}\mathbf{R}_{i-j} \\
\mathbf{a}_{\tau}^{n} &= \text{Masked-Softmax}(\mathbf{A}_{\tau}^{n})\mathbf{v}_{\tau}^{n} \\
\mathbf{o}_{\tau}^{n} &= \text{LayerNorm}(\text{Linear}(\mathbf{a}_{\tau}^{n}) + \mathbf{h}_{\tau}^{n-1}) \\
\mathbf{h}_{\tau}^{n} &= \text{Positionwise-Feed-Forward}(\mathbf{o}_{\tau}^{n})
\end{aligned}$$

10. MASS

Researchers from Microsoft Asia Research Institute put forward a new universal pre training method, mass, on ICML 2019, which comprehensively surpasses Bert and GPT in sequence to sequence natural language generation tasks.

Bert usually trains only one encoder for natural language understanding, while GPT's language model usually trains one decoder. If you want to use Bert or GPT in the natural language generation task of sequence to sequence, usually only separate the pre training encoder and decoder, so the encoder attention decoder structure is not jointly trained, and the memory mechanism will not be pre trained, and the attention mechanism of decoder to encoder is very important in this kind of task, so Bert and GPT can only achieve in this kind of task To sub optimal effect.

Aiming at the task of sequence to sequence natural language generation, Microsoft Asia Research Institute proposed a new pre training method: mass: masked sequence to sequence pre training. Mass randomly blocks a continuous segment of K length from the sentence, and then generates the segment by the encoder attention decoder model.



As shown in the figure above, the third to sixth words on the encoder side are shielded, and then the decoder side only predicts these consecutive words, while the other words are shielded. In the figure, "_" represents the shielded words.

Mass pre training has the following advantages:

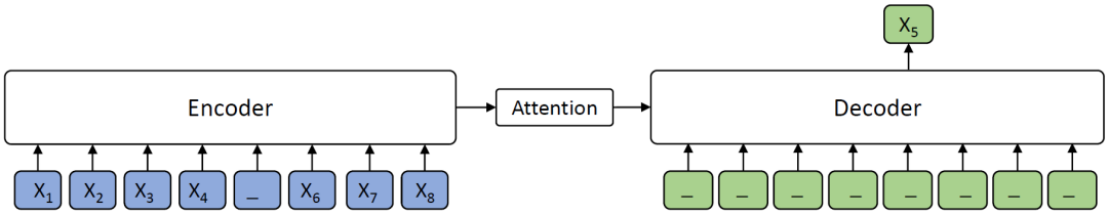
1. Other words on the decoder side (words that are not shielded on the encoder side) are shielded to encourage the decoder to extract information from the encoder side to help continuous segment prediction, which can promote the joint training of encoder attention decoder structure;
2. In order to provide more useful information for the decoder, the encoder is forced to extract the semantics of the unshielded words, so as to improve the

ability of the encoder to understand the source sequence text;

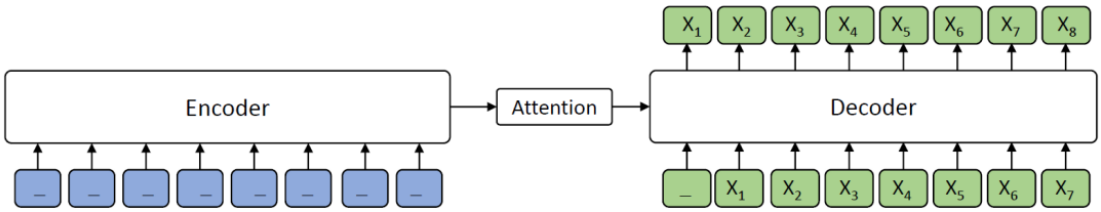
3. In order to improve the language modeling ability of the decoder, the continuous sequence segments are predicted by the decoder.

Another import point is Unified pre training framework.

Mass has an important super parameter K (the length of the continuous block of shielding). By adjusting the size of K, mass can include the shielding language model training method in Bert and the standard language model pre training method in GPT, which makes mass a general pre training framework.



When k = 1, according to the setting of mass, the encoder side shields a word and the decoder side predicts a word, as shown in the following figure. When there is no input information in decoder, the pre training method of masked language model in mass and Bert is equivalent



When k = m (M is the sequence length), according to the setting of mass, the encoder blocks all words and the decoder predicts all words. As shown in the figure below, because all words at the encoder end are blocked, the attention mechanism of the decoder is equivalent to not getting information. In this case, mass is equivalent to the standard language model in GPT.

The experiment analyzes the effect of pre training with different segment length k in the shielding mass model, as shown in the figure below.

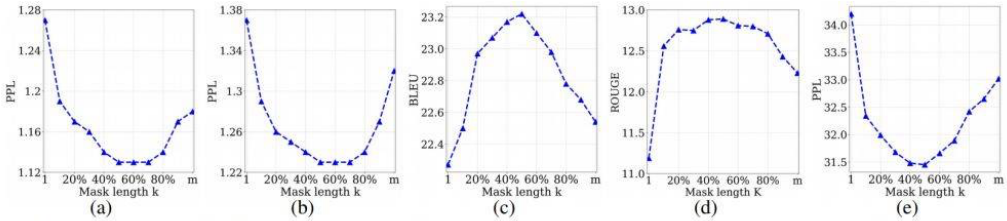


Figure 5. The performances of MASS with different masked length k, in both pre-training and fine-tuning stages, which include: the PPL of the pre-trained model on English (Figure a) and French (Figure b) sentences from WMT newstest2013 on English-French translation; the BLEU score of unsupervised English-French translation on WMT newstest2013 (Figure c); the ROUGE score (F1 score in RG-2) on the validation set of text summarization (Figure d); the PPL on the validation set of conversational response generation (Figure e).

When k is about half of the sentence length (50% m), the downstream task can achieve the optimal performance.

Pre-training:

Mass only needs unsupervised monolingual data (such as WMT news crawl data, Wikipedia data, etc.) for pre training. Mass supports cross language sequence to sequence generation (such as machine translation) and single language sequence to sequence generation (such as text summary generation and dialogue generation).

11. MT-DNN KD

MT-DNN is a multi-task deep neural network model released by Microsoft for learning universal language embeddings.

For MTL (Multi-task Learning), there are two advantages: 1) it makes up for the lack of data for some tasks; 2) it has a regular effect to prevent the model from overfitting.

The author of the paper believes that MTL and pretrain have a good complementary effect, so can we combine them to play the role of the two. To be more specific, first use BERT for pretrain and then MTL for finetune, which forms MT-DNN. It can be seen that the difference from BERT lies in the process of finetune, here MTL is used as the target.

Think about it from another aspect. In fact, when BERT did not come out, the

researcher was directly training the MTL model. Now that BERT is out, then why not try initializing with this?

a. Multitasking

MT-DNN is a combination of four types of NLU tasks: single sentence classification, sentence pair classification, text similarity score and relevance ranking. Here are some examples in GLUE:

Single sentence classification: For example, CoLA judges whether the sentence is grammatically appropriate and SST-2 judges the emotion of the sentence.

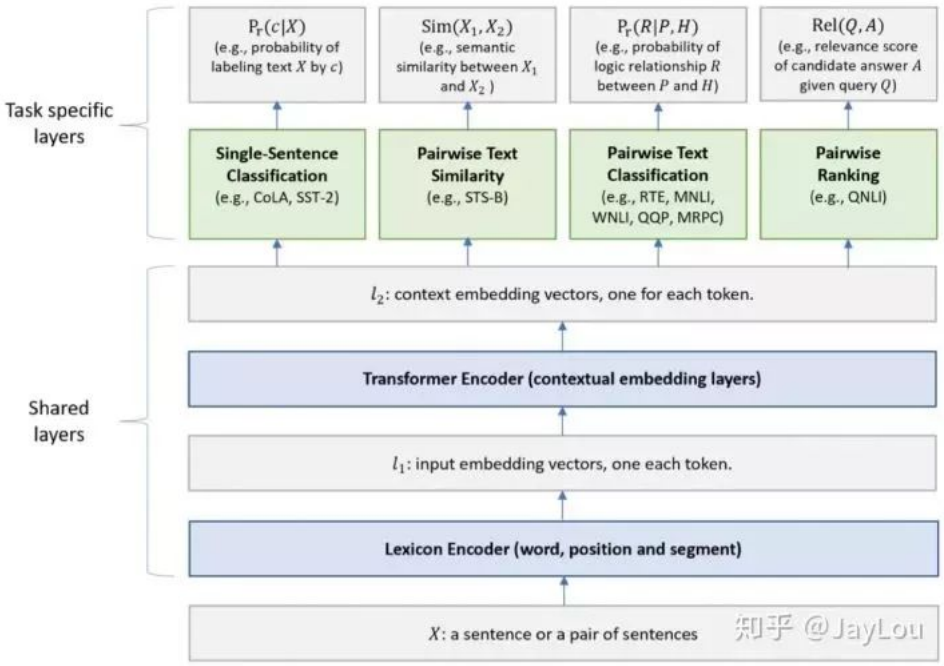
Text similarity: For example, STS-B scores the similarity of two sentences.

Sentence classification: For example, RTE and MNLI are text implied tasks, and QQP and MRPC are used to determine whether the two sentences are semantically consistent.

Relevance ranking: For example, QNLI, which was actually defined as a binary classification problem in the original GLUE task, but this paper defines it as a ranking problem, sampling a bunch of negative samples by itself, and then using softmax to learn ranking

Moreover, MTDNN introduce multi-tasks mechanism in the downstream tasks, unlike ERINE which introduce during the pre-training process.

b. Model structure



a) Classification of single sentences

Using the characterization of [CLS] as the feature and setting it to x, for the classification task of a single sentence, you can directly access a classification layer at behind, taking the SST-2 task as an example:

$$P_r(c|X) = softmax(W_{SST}^T \cdot x)$$

Loss function: Cross entropy

$$-\sum_c I(X, c) log(P_r(c|X))$$

b) Sentences similarity:

Using STS-B task as an example, pack two sentences together as input.

$$Sim(X_1, X_2) = sigmoid(w_{STS}^T \cdot x)$$

Loss function: MSE

$$(y - Sim(X_1, X_2))^2$$

c) Sentence pair classification

Take the NLI task as an example, here is a SAN network, a network that performs better in this task.

SAN calculating process:

a. Input premise $P = (p_1, ..., p_m)$ and hypothesis $H = (h_1, ..., h_n)$

b. Use BERT to get the representations of premise and hypothesis, $M^p = \mathbb{R}^{d*m}$ and $M^h = \mathbb{R}^{d*n}$

c. Start k steps reasoning: the initial state is self-attention of $M^h = \mathbb{R}^{d*n}$, $s^0 = \sum_j \alpha_j M_j^h$ and $\alpha_j = \frac{exp(w_1^T \cdot M_j^h)}{\sum_i exp(w_1^T \cdot M_i^h)}$.

For step k, $s^k = GRU(s^{k-1}, x^k)$ and $x^k = \sum_j \beta_j M_j^p, \beta_j = softmax(s^{k-1} W_2^T M^p)$.

d. The last layer is classification $P_r^k = softmax(W_3^T[s^k; x^k; |s^k - x^k|; s^k \cdot x^k])$.

e. At last, calculate the average of reasoning output: $P_r = avg([P_r^0, P_r^1, ..., P_r^{K-1}])$

d) Relevance ranking

Take QNLI as an example, here is mainly to calculate the similarity between two sentences

$Rel(Q, A) = g(w_{QNLI}^T \cdot x)$

Loss function: Sorting loss:

$$-\sum_{Q,A^+} P_r(A^+|Q)$$

$$P_r(A^+|Q) = \frac{exp(\gamma Rel(Q, A^+))}{\sum_{A' \in A} exp(\gamma Rel(Q, A'))}$$

c. Training process:

Algorithm 1: Training a MT-DNN model.

Initialize model parameters Θ randomly.
Pre-train the shared layers (i.e., the lexicon encoder and the transformer encoder).
Set the max number of epoch: $epoch_{max}$.
//Prepare the data for T tasks.
for t *in* $1, 2, ..., T$ **do**
  Pack the dataset t into mini-batch: D_t.
end
for $epoch$ *in* $1, 2, ..., epoch_{max}$ **do**
  1. Merge all the datasets:
   $D = D_1 \cup D_2 ... \cup D_T$
  2. Shuffle D
  for b_t *in* D **do**
    //b_t is a mini-batch of task t.
    3. Compute loss : $L(\Theta)$
      $L(\Theta)$ = Eq. 6 for classification
      $L(\Theta)$ = Eq. 7 for regression
      $L(\Theta)$ = Eq. 8 for ranking
    4. Compute gradient: $\nabla(\Theta)$
    5. Update model: $\Theta = \Theta - \epsilon\nabla(\Theta)$
  end
end

d. Experiments' performance:

Model	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	WNLI 634	AX	**Score**
BiLSTM+ELMo+Attn	36.0	90.4	84.9/77.9	75.1/73.3	64.8/84.7	76.4/76.1	79.9	56.8	65.1	26.5	70.5
Singletask Pretrain Transformer	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1/81.4	88.1	56.0	53.4	29.8	72.8
GPT on STILTs	47.2	93.1	87.7/83.7	85.3/84.8	70.1/88.1	80.8/80.6	87.2	69.1	65.1	29.4	76.9
$BERT_{LARGE}$	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7/85.9	91.1	70.1	65.1	39.6	80.4
MT-DNN	**61.5**	**95.6**	**90.0/86.7**	**88.3/87.7**	**72.4/89.6**	**86.7/86.0**	**98.0**	**75.5**	**65.1**	40.3	**82.2**

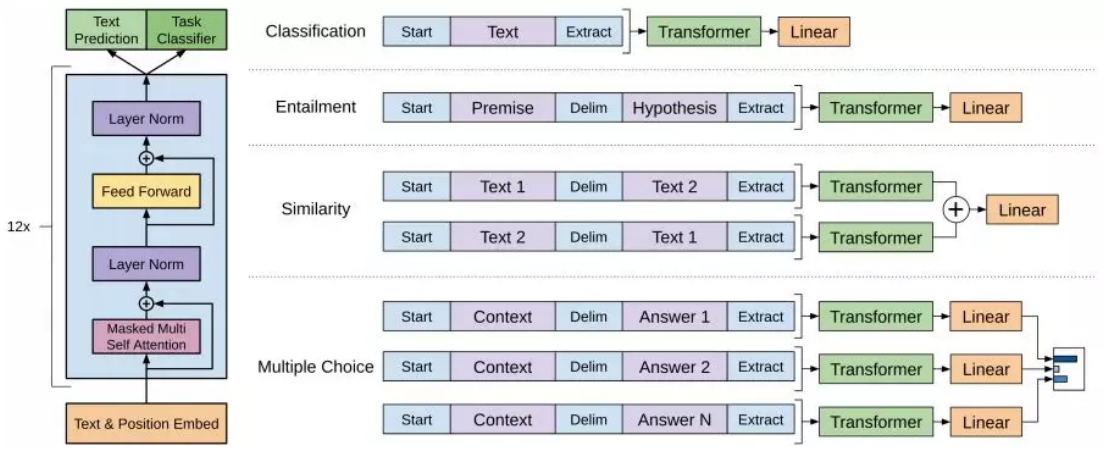
12. GPT

GPT is a semi supervised learning method, which is committed to using a large

number of dimensionless data to let the model learn "common sense", in order to alleviate the problem of insufficient annotation information. The specific method is to use the unlabeled data pre training model Pretrain before training the fine tune for labeled data, and to ensure that the two kinds of training have the same network structure.

GPT is also based on the transformer model, which is different from the transformer model for translation tasks: it only uses multiple Decker layers.

The following figure shows how to use the model to adapt to multi classification, text implication, similarity and multi selection without modifying the main structure of the model.



Classification task: input is the text, and the vector of the last word is directly used as the fine-tuning input to get the final classification result (multiple classifications are allowed)

Reasoning task: input is a priori + separator + hypothesis, and the vector of the last word is directly used as the input of fine-tuning to get the final classification result, that is, whether it is true or not

Sentence similarity: input is that two sentences are reversed, the vectors of the last word are added, and then linear is used to get the final classification result, that is, whether they are similar or not

Q & a task: input is to put context and questions together and separate with multiple answers, and the middle is also separated by separators. For each answer, the vector of the last word of the sentence is used as fine-tuning input, and then linear is used, and the result of multiple linear is used softmax to get the highest final probability

For the Q & a task, how to conduct softmax for the results of the last multiple linear?

For Q & A tasks, a question corresponds to multiple answers, and finally I want to take the most accurate answer (with the highest score) as the result. I can unify the dimensions by doing transformer for multiple answers, and then make linear respectively, and then make softmax for multiple linear. Before, I used to do softmax for one linear, but now I can directly take the maximum probability value How to implement softmax with multiple linear?

The above is the general description of GPT. Using unsupervised pre training and supervised fine-tuning can achieve most NLP tasks, and the effect is significant, but it is not as good as Bert. However, GPT uses one-way transformer to solve the text generation task that Bert cannot solve.

Model implementation

In the pre training part, u is used to represent each token. When the window length is set to K and the ith word in the prediction sentence is set, the K word before the ith word is used. At the same time, the I word is predicted according to the super parameter 0. In short, use the words in front to predict the words in the back.

$$L_1(u) = \sum_i \log P(u_i \mid u_{i-k}, \ldots, u_{i-1}; \Theta)$$

In the final tune part of the supervised training, for example, the sentence that judges the emotional color of the sentence (dichotomous problem) contains M words X1 XM, adding a full connection layer after the model trained by pretain, is used to learn the parameter WY describing the relationship between input information X and target y, and finally predict target y.

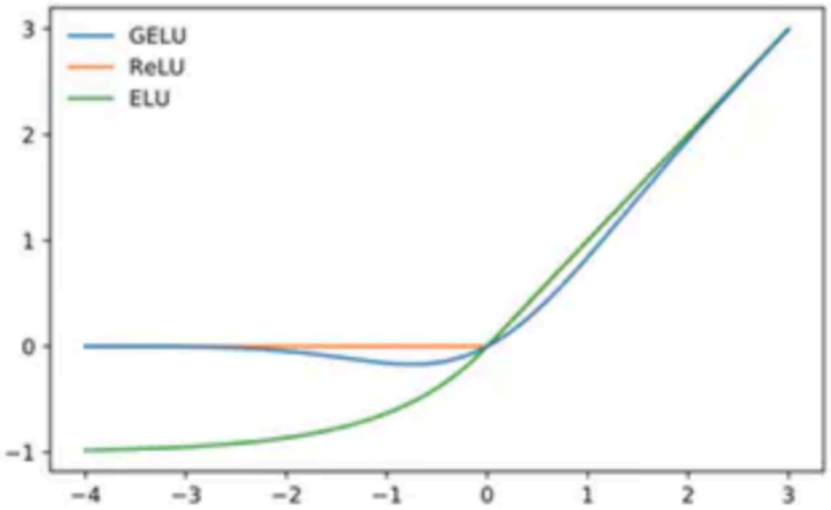
$$P(y \mid x^1, \ldots, x^m) = \text{softmax}(h_l^m W_y)$$

$$L_2(C) = \sum_{(x,y)} \log P(y \mid x^1, \ldots, x^m)$$

Considering the L1 and L2 in the above formula, add the weight parameter λ to control its proportion and calculate L3 as the basis of optimization.

$$L_3(C) = L_2(C) + \lambda * L_1(C)$$

Compared with the basic transformer, GPT also has the following modifications: Taking glue (Gaussian error linear unit) as the error function, glue can be regarded as the improvement method of relu. Relu will convert the data less than 1 to 0, and the part greater than 1 will remain unchanged, while Gelu will slightly adjust it, as shown in the following figure:



For position coding, the basic transformer uses sine cosine function to construct position information, which does not need to train corresponding parameters; while GPT uses absolute position information as coding.

13. GPT-2

brief introduction

The core idea of gpt2 is that we can use unsupervised pre training model to do supervised tasks.

Gpt-2 wants to model without understanding words at all, so that the model can handle any coding language. Gpt-2 is mainly aimed at the zero shot problem. It has a great improvement in solving a variety of unsupervised problems, but it is worse for supervised learning.

The effect comparison between unsupervised learning and supervised learning is like two children learning, one is broad reading, but the reading is not necessarily

the test; the other is specialized in the test site, and fixed-point optimization. The result is that one is better in the exam, the other is more capable and can solve all kinds of problems, especially for the problems without definite answers. They have their own advantages in different fields.

At present, gpt-2 can be used to generate answers in translation, question and answer, reading comprehension, summary and other fields of answering with words, and the most popular one is the continuous story model

Model implementation

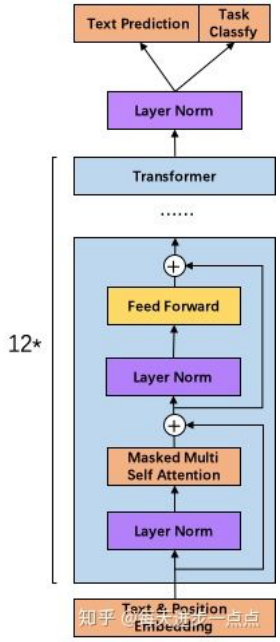
Gpt-2 still follows the pattern of GPT one-way transformer, but makes some improvements and changes. What are the differences between gpt-2 and GPT? Look at the following:

1. Gpt-2 removes the fine tuning layer: instead of fine tuning modeling for different tasks, it does not define what tasks this model should do, and the model will automatically identify what tasks need to be done. It's like a person who has a wide range of books. You can ask him any kind of questions at his fingertips. Gpt-2 is such a wide range of books model.

2. Add data set: since we need to read widely, we need to have books first, so gpt-2 has collected a wider and more quantitative corpus to form the data set. The dataset contains 8 million Web pages, 40 gigabytes in size. Of course, these data sets are high-quality text after filtering, so the effect can be better~

3. Add network parameters: gpt-2 increases the number of layers of transformer stack to 48, the dimension of hidden layer is 1600, and the number of parameters is up to 1.5 billion. What's the concept of 1.5 billion? The parameter value of Bert is only 300 million. Of course, such parameter value doesn't mean that anyone can achieve it. It depends on the amount of money~

4. Adjust transformer: put layer normalization before each sub block, and add another layer normalization after the last self attention. This feeling in the paper is ambiguous. If only you could give a picture. However, you can understand this detail through the code. The figure below is a schematic diagram of how to add layer normalization for your reference



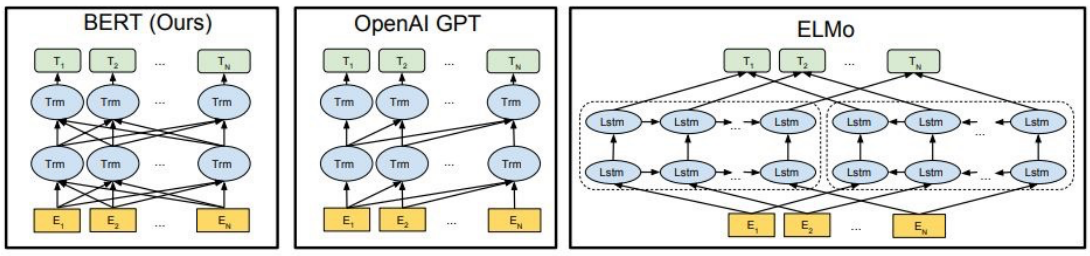
5. Others: gpt-2 increased the number of vocabularies to 50257; the maximum context size increased from 512 of GPT to 1024 tokens; batchsize increased to 512.

14. BERT

1.BERT model

The full name of Bert is bidirectional encoder representation from transformers, that is, the encoder of bidirectional transformer, because the decoder cannot obtain the information to be predicted. The main innovation of the model is the pre train method, that is to say, two methods, masked LM and next sentence prediction, are used to capture the representation of words and sentences respectively.

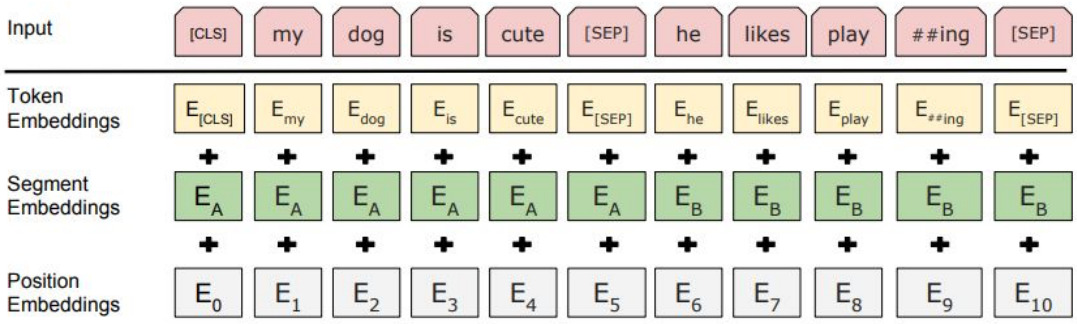
1.1 model structure



$$P(w_i|w_1,\ldots,w_{i-1},w_{i+1},\ldots,w_n)$$

1.2 Embedding

Embedding here consists of three kinds of embedding summation:



Among them:

• token embeddings is a word vector, the first word is the CLS flag, which can be used for subsequent classification tasks

• segment embeddings is used to distinguish two sentences, because pre training not only does LM, but also does classification task with two sentences as input

• position embeddings is different from the transformer in the previous article. It's not a trigonometric function, but a learned one

1.3 Pre-training Task 1#: Masked LM

MLM can be understood as cloze. The author will randomly mask 15% of the words in each sentence and use their context to make predictions

Here, hairy is masked, and then the unsupervised learning method is used to predict what the words in the mask position are. However, there is a problem in this method, because 15% of the words in the mask have a high number, which will lead to some words never seen in the fine tuning stage. In order to solve this problem, the author has done the following processing:

• 80% of the time is using [mask], my dog is hairy → my dog is [mask]

• 10% of the time is to randomly choose a word to replace mask's word, my dog is hairy - > my dog is apply

• my dog is hairy - > my dog is hairy

So why use random words with a certain probability? This is because the

transformer needs to keep the distributed representation of each input token, otherwise the transformer will probably remember that this [mask] is "hairy". As for the negative impact of using random words, it is explained that all other tokens (i.e. non "hairy" tokens) share a probability of 15% * 10% = 1.5%, and the impact is negligible. The global visibility of transformer increases the acquisition of information, but does not allow the model to obtain the full amount of information.

Be careful:

• there is a parameter "dupe" factor to determine the number of data duplicates.

• where the create - instance - from - document function is a sample of a sentence pair constructed. For each sentence, Mr. Cheng [CLS] + A + [SEP] + B + [SEP], long (0.9) and short (0.1), plus mask, and then make a sample object.

• the tokens returned by the create - masked - LM - predictions function are those that have been replaced by occluded words

• masked - LM - labels are the real labels corresponding to the positions of the occluded words.

1.4 Pre-training Task 2#: Next Sentence Prediction

Select some sentence pairs a and B, of which 50% data B is the next sentence of a, and the remaining 50% data B is randomly selected in the corpus. Learn the correlation among them. The purpose of adding such pre training is that many NLP tasks such as QA and NLI need to understand the relationship between the two sentences, so that the pre training model can better adapt to such tasks.

2. advantages and disadvantages

2.1 advantages

• transformer encoder has self attention mechanism, so Bert has two-way function

• due to the influence of two-way function and multi-layer self attention mechanism, Bert must use the language model masked LM of close version to complete token level pre training

In order to obtain the semantic representation of sentence level higher than words, Bert added the next sentence prediction to do joint training with masked LM

In order to adapt to multi task migration learning, Bert designed a more general

input layer and output layer

• small fine tuning costs

2.2 disadvantages

• the random occlusion strategy of Task1 is a little rough. It is recommended to read "data nosing as smoothing in neural network language models"

• [mask] markers will not appear in the actual prediction, and too much [mask] is used in training to affect the performance of the model;

• only 15% of tokens per batch are predicted, so Bert converges slower than the left to right model (they predict each token)

• Bert consumes a lot of hardware resources (16 TPUs for a larger model, which takes four days; 64 TPUs for a larger model, which takes four days).

15. RoBERTa

A robust optimized Bert, a powerful optimized Bert method

In terms of model scale, calculation power and data, compared with Bert, it has the following improvements:

1. Larger model parameters (according to the training time provided in the paper, the model used 1024 V100 GPUs to train for one day)

2. Bigger bath size. Roberta used a larger bath size during training. Tried to use the bacth size from 256 to 8000.

3. More training data (including 160GB plain text including cc-news, etc.). While the original Bert trained with 16GB bookcorpus data set and English Wikipedia)

Roberta has the following improvements in training methods:

1. Remove the next prediction (NSP) task

2. Dynamic mask. Bert relies on random mask and prediction token. The original Bert implementation performs a mask once during data preprocessing to get a static mask. Roberta uses a dynamic mask: a new mask pattern is generated each time a sequence is entered into the model. In this way, in the process of massive data input, the model will gradually adapt to different mask strategies and learn different language representations.

3. Text encoding. Byte pair encoding (BPE) is a mixture of character level and word level representations, which supports the processing of many common words in natural language corpus. The original Bert implementation uses character level BPE vocabulary with a size of 30K, which is learned after preprocessing the input with heuristic word segmentation rules. Instead of using this approach, Facebook researchers consider training Bert with a larger byte level BPE vocabulary, which contains 50K subword units, without any additional preprocessing or segmentation of input.

Training process analysis

This section discusses which quantitative indicators affect the pre training Bert model while keeping the model architecture unchanged. First, keep the training Bert model architecture unchanged, and its configuration is the same as that of Bert base (L = 12, H = 768, a = 12110m parameters).

4.1 Static vs Dynamic Masking

Original static mask:

In Bert, when preparing training data, each sample will only be randomly masked once (so each epoch is repeated), and the same mask will be used for each subsequent training step, which is the original static mask, that is, a single static mask, which is the original Bert's practice.

Modified static mask:

During the preprocessing, the dataset is copied 10 times, and each copy uses a different mask (40 epochs in total, so the data corresponding to each mask is trained with 4 epochs). This is equivalent to using 10 kinds of static masks to train 40 epochs in the original data set.

Dynamic mask:

Instead of performing a mask during preprocessing, a mask is generated dynamically each time an input is provided to the model, so it changes all the time.

The experimental results of different modes are shown in the table below. Where reference is the original static mask used by Bert, and static is the modified static mask.

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Table 1: Comparison between static and dynamic masking for $\text{BERT}_{\text{BASE}}$. We report F1 for SQuAD and accuracy for MNLI-m and SST-2. Reported results are medians over 5 random initializations (seeds). Reference results are from Yang et al. (2019).

It can be seen from table 1 that the static mask of the modified version is equivalent to the original static mask of Bert; the dynamic mask is similar to the static mask, or slightly better.

Based on the judgment of the above results and the advantages of dynamic mask in efficiency, the following experiments in this paper adopt dynamic mask.

4.2 Model Input Format and NSP

The original Bert contains two tasks: predicting the words dropped by the mask and the next prediction. In view of recent studies (lamp and conneau, 2019; Yang et al., 2019; Joshi et al., 2019), which begin to question the necessity of the next sentence prediction (NSP), this paper designs the following four training methods:

SEGMENT-PAIR + NSP：

The input consists of two parts, each part is a segment from the same document or a different document (segments are consecutive sentences), and the total number of tokens of the two segments is less than 512. Pre training includes MLM tasks and NSP tasks. This is the original Bert approach.

SENTENCE-PAIR + NSP：

The input also includes two parts, each part is a single sentence from the same document or different documents, and the total number of tokens of these two sentences is less than 512. Since these inputs are significantly less than 512 tokens, increase the size of batch size to keep the total number of tokens similar to that of segment-pair + NSP. Pre training includes MLM tasks and NSP tasks.

FULL-SENTENCES：

The input has only one part (not two parts), and consecutive sentences from the same document or different documents. The total number of tokens does not exceed 512. The input may cross the document boundary, and if it does, a document boundary token is added at the end of the previous document. Pre training does not include NSP tasks.

DOC-SENTENCES：

The input has only one part (not two parts). The structure of input is similar to full-sessions, but it does not need to cross the document boundary. Its input comes from consecutive sentences of the same document, and the total number of tokens does not exceed 512. The input sampled near the end of the document can be less than 512 tokens, so in these cases, increase the batch size dynamically to reach the same total number of tokens as full-sessions. Pre training does not include NSP tasks.

The comparison results of the above four training modes are shown in Table 2:

Model	**SQuAD 1.1/2.0**	**MNLI-m**	**SST-2**	**RACE**
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

Table 2: Development set results for base models pretrained over BOOKCORPUS and WIKIPEDIA. All models are trained for 1M steps with a batch size of 256 sequences. We report F1 for SQuAD and accuracy for MNLI-m, SST-2 and RACE. Reported results are medians over five random initializations (seeds). Results for $\text{BERT}_{\text{BASE}}$ and $\text{XLNet}_{\text{BASE}}$ are from Yang et al. (2019).

Bert uses the input format of segment-pair (can contain multiple sentences). From the experimental results, if NSP loss is used, segment-pair is better than that of sentence-pair (two sentences). Finding a single sentence will damage the performance of the downstream task, which may be the reason why the model can't learn remote dependency. The next comparison is to compare the NSP free training with the training from a single document (DOC sentence) block of text. We found that compared with Devlin et al. (2019), the performance of this setting is better than that of the original Bert base result: eliminating NSP loss can be

equal to or slightly higher than the original Bert in the performance of downstream tasks. Possible cause: the original Bert implementation only removes the loss items of NSP, but still keeps the input form of segment-pari.

Finally, the experiment also found that the performance of restricting sequence to single document is slightly better than that of sequence from multiple documents. However, in the doc-settings policy, the sample at the end of the document may be less than 512 tokens. In order to keep the total number of tokens in each batch at a high level, the batch size needs to be adjusted dynamically. For the convenience of processing, doc-settings input format is adopted later.

4.3 Training with large batches

Previous neural machine translation studies have shown that when using very large Mini bats for training, properly improving the learning rate can not only improve the optimization speed, but also improve the final task performance. Recent research shows that Bert can also receive large batch training. Devlin et al. (2019) initially trained Bert base with only one million steps, and the batch size was 256 sequences. Through gradient accumulation, 125k steps of batch size = 2K or 31k steps of batch size = 8K are trained, which are approximately equivalent in calculating cost.

In Table 3, we compare the perplexity (confusion, an indicator of the language model) and the final task performance of Bert basee when increasing the batch size. It can be observed that large batches training improves the confusion of the masked language modeling target and the accuracy of the final task. Large batches is also easier for distributed data parallel training. In the following experiments, the text uses bacth size = 8K for parallel training.

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

4.4 Text Encoding

Byte pair encoding (BPE) (sennrich et al., 2016) is a mixture of character level and word level representations. This encoding scheme can deal with a large number of common words in natural language corpus. BPE does not rely on complete words, but on sub word units. These sub word units are extracted by statistical analysis of the training corpus, and their word table size is usually between 10000 and 100000. When modeling massive and diverse corpus, Unicode characters occupy the majority of the vocabulary. Radford et al. (2019) introduced a simple but efficient BPE, which uses byte pairs instead of Unicode characters as sub word units.

The following two BPE implementation methods are summarized:

Based on char level: the original Bert method, it is obtained by heuristic word stemming and post-processing of the input text.

Based on bytes level: the difference with char level is that bytes level uses bytes instead of Unicode characters as the basic unit of sub word, so any input text can be encoded without introducing the unkown tag.

When BPE of bytes level is adopted, the vocabulary size increases from 30000 (char level of original BERT) to 50000. This adds 15 million and 20 million additional parameters to Bert base and Bert large, respectively.

Previous studies have shown that this approach can cause slight performance degradation on some downstream tasks. But the authors believe that the

advantages of this unified coding will outweigh the slight performance degradation. And the author will further compare different encoding schemes in the future work.

16. DistilBERT

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

On the basis of Bert, distilbert is a miniaturized Bert trained by knowledge distillation technology. On the whole, this paper is very simple, but introduces knowledge distillation technology to train a small Bert. The specific methods are as follows:

1) The original Bert base is given as the teacher network.

2) On the basis of Bert base, the number of network layers is halved (that is, from the original 12 layers to 6 layers).

3) Using the soft tag of teacher and the hidden layer parameters of teacher to train student network.

The loss function in training is defined as the linear sum of three kinds of loss functions, which are respectively:

1） LceLce。 This is the cross entropy of the probability distribution of the output of the softmax layer of the teacher network and that of the output of the softmax layer of the student network.

2） $Lmlm$Lmlm。 This is the probability distribution of the output of the softmax layer of the student network and the cross entropy of the real one hot tag

3） $Lcos$Lcos。 This is the cosine similarity value of student network hidden layer output and teacher network hidden layer output. In the above, we say that the number of student network layers is only 6, and the number of teacher network layers is 12. Therefore, I think that when calculating the loss, the first layer of student is used to correspond to the second layer of teacher, the second layer of student is used to correspond to the fourth layer of teacher, and so on.

The author has done some work on the initialization of students. The author uses the parameters of teacher to initialize the network parameters of students. The method is similar to the above. The second layer of teacher is used to initialize the

first layer of students and the fourth layer of teacher is used to initialize the second layer of students.

The author also explains why to reduce the number of layers of the network without reducing the size of the hidden layer. In the framework of modern linear algebra, in tensor calculation, reducing the last dimension (i.e. the size of the hidden layer) has little effect on the calculation efficiency, but on the contrary, it reduces the number of layers and improves the calculation efficiency.

In addition, the author removed sentence vector and Pooler layer here, and did not see the loss function of NSP task, so I think the author also removed NSP task (mainly many people proved that the task had no effect).

On the whole, although the method is simple, the effect is very good. The model size is reduced by 40% (66m), the inference speed is increased by 60%, but the performance is only reduced by about 3%.

17. ALBERT

Albert mainly makes changes from the model architecture, which can greatly reduce the size of the model, but does not improve the inference speed. Albert has made three major changes: factorized embedding parameterization, cross layer parameter sharing, inter sense coherence loss. Let's start with three changes:

Factorized embedding parameterization

In Bert, the embedded word vector size e of embedding layer is equal to the hidden layer size h, but the author thinks that the embedding layer only embeds words. In this layer, words are independent from each other, and the information contained in the vectors after embedding words is only the information of the current words, so the vector length does not need to be that large (the commonly used word2vec vector length is generally not more than But because the hidden layer will calculate self attention with other words, the vector corresponding to the hidden layer words contains context information. At this time, the information contained is very rich. It is easy to lose information with a small vector, so it is necessary to set the size of the hidden layer a little larger. In view of such analysis, the author thinks that it is unreasonable to set the embedding layer and hidden

layer size to be equal in Bert (E = H = 768 in Bert base). And in the embedding layer, there is a large word embedding matrix V x E. In this case, V is vocab size (usually larger, more than 20000 in BERT). Therefore, when e is large, there are many parameters here. Based on the above analysis, e can not be set as large. Therefore, the author makes a matrix decomposition here, and decomposes the matrix V x H (E) into two small matrices V x E, e x h, e < < H. Instead of E = h, set E to a value much smaller than h, and then map the word vector dimension to h through a matrix e x H.

Cross-layer parameter sharing

In the transformer structure, we can also choose to share some parameters. It is better to share the parameters in self attention or FFN layer. Here, the author shares all the parameters in encoder (including self attention and FFN). Combined with the above matrix decomposition, the model parameters are greatly reduced. The specific results are as follows:

Model		Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
	xlarge	1270M	24	2048	2048	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	59M	24	2048	128	True
	xxlarge	233M	12	4096	128	True

Table 2: The configurations of the main BERT and ALBERT models analyzed in this paper.

In addition, the parameter sharing mechanism between layers also makes the L2 distance and cosine distance of input and output of each layer more smooth, as shown in the following figure:

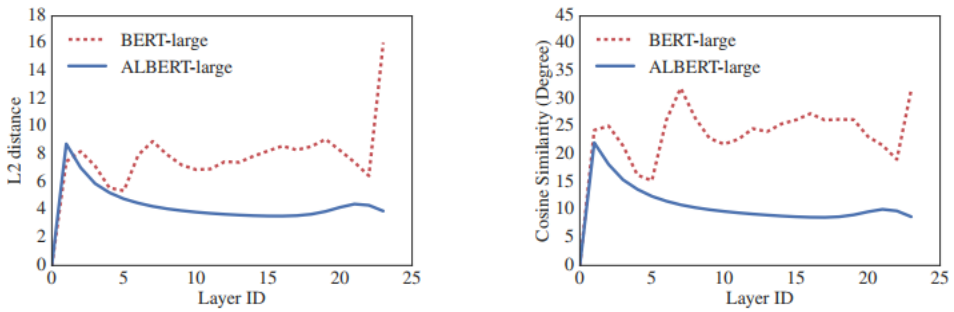


Figure 2: The L2 distances and cosine similarity (in terms of degree) of the input and output embedding of each layer for BERT-large and ALBERT-large.

Inter-sentence coherence loss

The above two changes are mainly to reduce the parameters of the model, while the changes here are mainly to improve the performance of the model on the downstream tasks. The author thinks that the NSP task in Bert is actually problematic. When designing this task, he hoped to learn sentence consistency, but in fact, when constructing sentence pairs, he chose sentences from different documents Therefore, the network does not need to learn the consistency of sentences. As long as we learn the topic of sentences (because different documents may have different topics), we can judge whether two sentences are negative examples. Therefore, the author thinks that NSP task only learns the topic of sentences, and topic classification is a shallow semantic NLP task.

So here, the author proposes a new task SOP (sentence order prediction). The positive example and Bert are constructed here, while the negative example is to reverse the order of the two sentences in the positive example. Here, the sentences in the negative example are all from the same document, so it is impossible to distinguish the positive example and the negative example through the topic of the sentence. It is necessary to understand the deep semantics of the sentence to distinguish The experimental results are also given

	Intrinsic Tasks			Downstream Tasks					
SP tasks	MLM	NSP	SOP	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
None	54.9	52.4	53.3	88.6/81.5	78.1/75.3	81.5	89.9	61.7	79.0
NSP	54.5	90.5	52.0	88.4/81.5	77.2/74.6	81.6	**91.1**	62.3	79.2
SOP	54.0	78.9	86.5	**89.3/82.3**	**80.0/77.1**	**82.0**	90.3	**64.0**	**80.1**

As shown in the above table, learning SOP (86.5) task can also achieve good results in NSP (78.9) task, but learning NSP (90.5) task cannot achieve results in SOP (52.0) task.

The above three points are changed in this paper, so matrix decomposition and parameter sharing reduce the model parameters a lot, but because the number of layers has not changed, the calculation amount has not decreased in the inference, so the inference speed has not improved, but the training speed has improved. The introduction of SOP task can also improve the effect of the model. In addition, the n-gram mask is used in the mask mode of Albert. In fact, it is the same as span mask in spanbert, except that span controls the maximum length of the mask to 10, while here it controls the maximum length to 3. In addition, lamb optimizer is used in the optimization algorithm.

18. TinyBERT

Tinybert also uses the method of knowledge distillation to compress the model, but in the design, it calls distillbert to do more work. The author puts forward two points: knowledge distillation for transformer structure and knowledge distillation for pre training and fine tuning.

Here, the author constructs four kinds of loss functions to constrain the parameters of each layer in the model to train the model. The specific structure of the model is as follows:

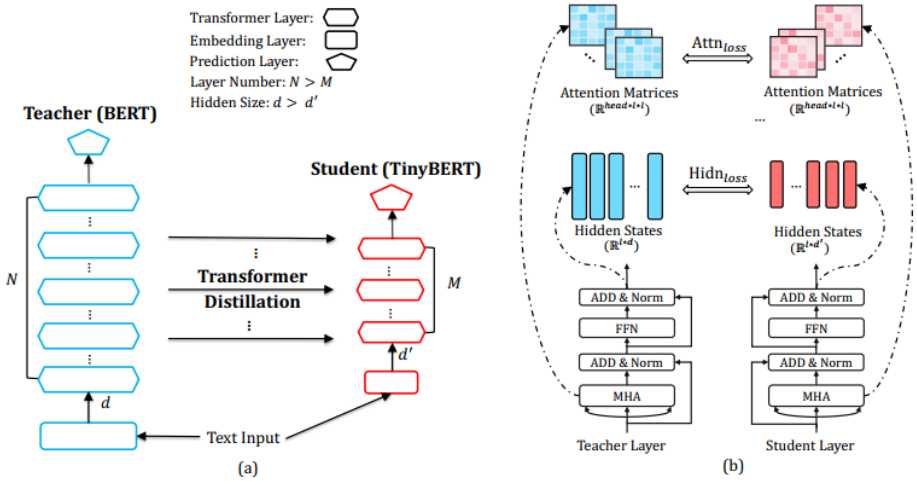


Figure 1: An overview of Transformer distillation: (a) the framework of Transformer distillation, (b) the details of Transformer-layer distillation consisting of Attn_{loss}(attention based distillation) and Hidn_{loss}(hidden states based distillation).

Four kinds of loss are constructed, which are embedding layer, attention weight matrix, hidden layer output and predict layer. This can be unified into a loss function:

$$\mathcal{L}_{\text{model}} = \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{\text{layer}}(S_m, T_{g(m)}),$$

Attention weight matrix

$$\mathcal{L}_{\text{attn}} = \frac{1}{h} \sum_{i=1}^{h} \text{MSE}(\boldsymbol{A}_i^S, \boldsymbol{A}_i^T),$$

Hidden layer output

$$\mathcal{L}_{\text{hidn}} = \text{MSE}(\boldsymbol{H}^S \boldsymbol{W}_h, \boldsymbol{H}^T),$$

Because the hidden layer size of the student network is usually set smaller than that of the teacher, in order to keep the same dimension in the calculation, we use a matrix $\boldsymbol{W}_h$ to map the hidden layer vector of the student to the same space as the teacher.

embedding layer

$$\mathcal{L}_{\text{embd}} = \text{MSE}(\boldsymbol{E}^S \boldsymbol{W}_e, \boldsymbol{E}^T),$$

predicy layer

$$\mathcal{L}_{\text{pred}} = -\text{softmax}(\boldsymbol{z}^T) \cdot \text{log\_softmax}(\boldsymbol{z}^S / t),$$

The prediction layer is also known as softmax layer. The loss function here is cross entropy, and t is the temperature parameter, which is set to 1 here.

The above four loss functions are the knowledge distillation methods proposed by the author for transformer. In addition, the author thinks that in addition to pre training distillation, the teacher's knowledge can also be used to train the model in fine tuning to achieve better results in downstream tasks. Therefore, the author proposes two stages of knowledge distillation, as shown in the following figure:

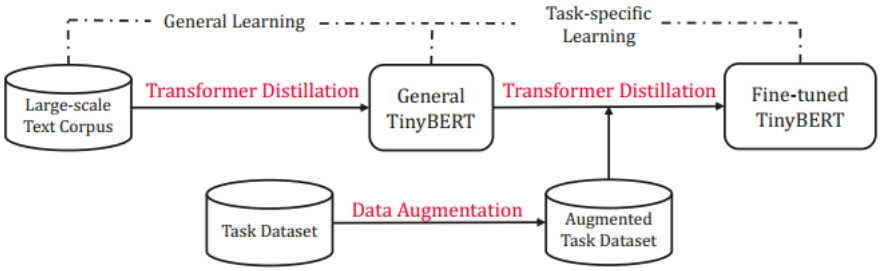


Figure 2: The illustration of TinyBERT learning

In essence, it is to distill a general tinybert in pre training, and then distill a fine tuned tinybert using task Bert on the basis of general tinybert.

19. SpanBERT

In this paper, the author proposes a new word segmentation level pre training method spanbert, which is better than Bert in the existing tasks, and has made great progress in word segmentation tasks such as Q & A and anaphora resolution. The Bert model is improved as follows:

A better span mask scheme is proposed. Spanbert no longer adds a mask to a random single token, but adds a mask to a random adjacency participle;

By adding the span boundary objective (SBO) training objective, we can predict the content of the word segmentation by using the representation of the word segmentation boundary, and no longer rely on the representation of a single token in the word segmentation, which enhances the performance of Bert, especially in some tasks related to span, such as extraction Q & A;

The result is similar to that of xlnet. It is found that it is better to train directly with a long sentence without adding the next sentence prediction (NSP) task.

Figure 1 shows the principle of the model.

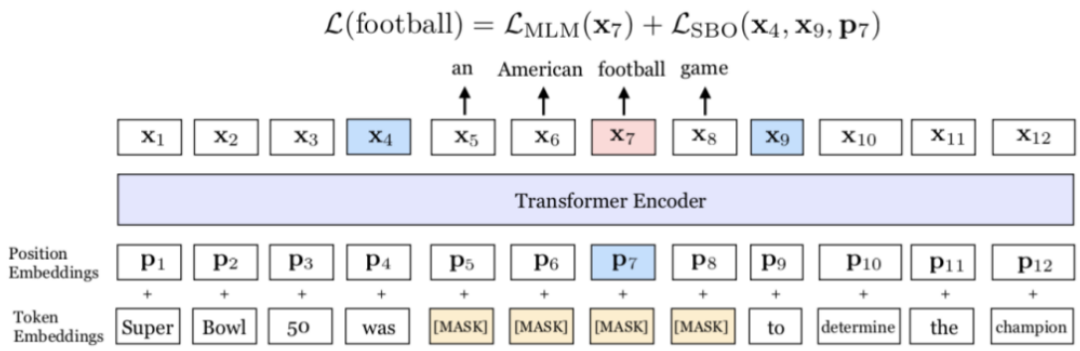


图1 SpanBERT 图示。在该示例中，分词 an American football game上添加了掩膜。模型之后使用边界词 was和 to来预测分词中的每个单词。

Participle mask

For each word sequence x = (x1,..., xn), the author selects the word by iteratively sampling the word segmentation of the text until the required size of the mask is reached (for example, 15% of x), and forms a subset y of X. In each iteration, the author first samples the length of the word segmentation from the geometric distribution L ~ geo (P). The geometric distribution is partial distribution, which tends to the shorter word segmentation. After that, the author randomly (evenly)

chooses the starting point of segmentation.

According to the geometric distribution, the length of a segment (span) is randomly selected first, then the starting position of the segment is randomly selected according to the uniform distribution, and finally the segment is covered according to the length. The author sets the geometric distribution as P = 0.2, and the maximum length of clipping can only be 10 (not more than 10, but should be discarded). Using this scheme, the average sampling length distribution can be obtained. So the average length of participle is 3.8. The author also measured the degree of word segmentation, which made the word segmentation longer. Figure 2 shows the distribution of word segmentation mask length.

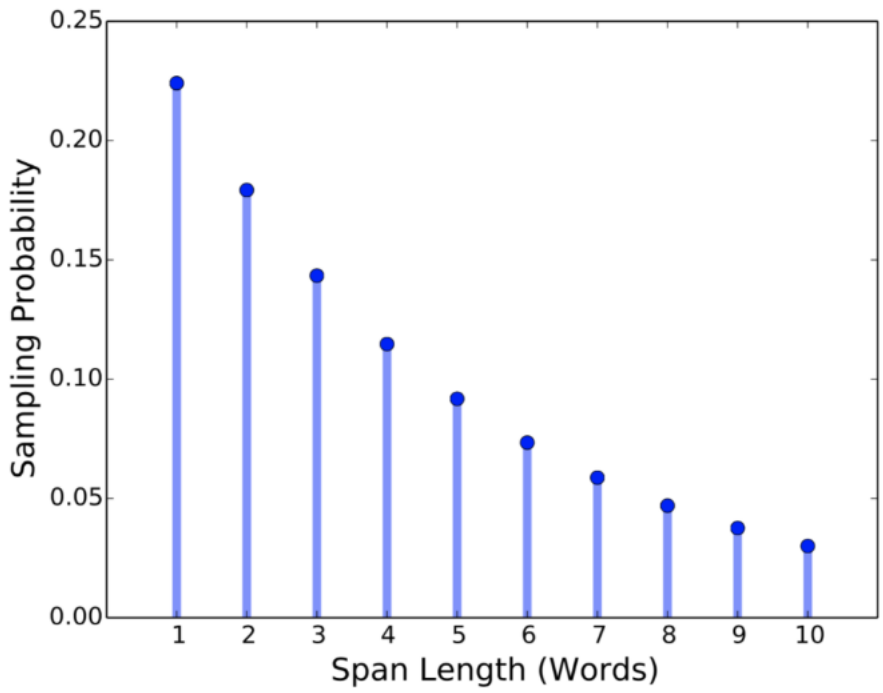


图2 分词长度（单词） https://blog.csdn.net/weixin_37947156

As in Bert, the author sets the scale of Y as 15% of X, 80% of which use [mask] to replace, 10% use random words to replace, and 10% remain unchanged. In contrast, the author makes this substitution at the word segmentation level, rather than replacing each word individually.

Segmentation boundary objective (SBO)

Segmentation selection model usually uses its boundary words to create a fixed length segmentation representation. In order to adapt to the model, the author

hopes that the sum of the end participle and the middle participle should be the same as much as possible. For this reason, SBO is introduced, which only uses the observed boundary words to predict the content of masked segmentation (Figure 1).

The specific method is to take two words of the front and back boundary of the span during training. It is worth noting that these two words are not in the span, and then use these two word vectors plus the position vector of the covered words in the span to predict the original words.

$$\mathbf{y}_i = f(\mathbf{x}_{s-1}, \mathbf{x}_{e+1}, \mathbf{p}_i)$$

The detailed method is to splice the word vector and position vector. The author uses a two-layer feedforward neural network as the representation function. The network uses the Gelu activation function, and uses the layer regularization:

$$\begin{aligned} \mathbf{h} &= \text{LayerNorm}(\text{GeLU}(W_1 \cdot [\mathbf{x}_{s-1}; \mathbf{x}_{e+1}; \mathbf{p}_i])) \\ f(\cdot) &= \text{LayerNorm}(\text{GeLU}(W_2 \cdot \mathbf{h})) \end{aligned}$$

The author uses vector representation Yi to predict Xi, and uses cross entropy as a loss function, which is the loss of SBO target, like MLM. After that, the loss is combined with the loss of Bert's masked language model (MLM) to train the model.

$$\mathcal{L}(\text{football}) = \mathcal{L}_{\text{MLM}}(\mathbf{x}_7) + \mathcal{L}_{\text{SBO}}(\mathbf{x}_4, \mathbf{x}_9, \mathbf{p}_7)$$

Single sequence training

There is another difference between spanbert and the original Bert training. Instead of using the next sentence prediction (NSP) task, spanbert directly uses single sequence training, that is, it does not add NSP task at all to judge whether two sentences are the upper and lower sentences, and directly uses one sentence to train. The author speculates that the possible reasons are as follows: (a) the longer context is more favorable for the model, and the model can obtain the longer context (similar to part of the effect of xlnet); and (b) adding the context information of another text will bring noise to the MLM language model.

Therefore, spanbert does not use the NSP task. It only samples a single adjacent

segment, which has a maximum length of 512 words. Its length is the same as the sum of the maximum length of the two segments used by Bert. Then MLM plus SBO task is used for pre training.

The main training details are:

In training, dynamic masking is used instead of the mask used by Bert in preprocessing;

The strategy of canceling random sampling short sentences in Bert;

There are also some parameter changes in the Adam optimizer.