Unknown Title

: 25/9/2021

What Is Pre-Training in NLP? Introducing 5 Key Technologies

Part 1: Building your understanding of pre-training AI

dan lee Follow
Feb 24, 2020 · 8 min read ★

Top highlight

What Is Pre-Training in NLP? Introducing 5 Key Technologies

Part 1: Building your understanding of pre-training AI

Welcome back to my blog for engineers who want to learn AI!

Starting with this post, we'll be launching into a new series of articles on pre-training in NLP. Today, we'll begin by forming a big picture.

In addition to defining pre-training, this article will give you a bird's eye view of its development by discussing the main technologies at play. In the articles to follow, we'll discuss each one in detail.

By the end of this post, you should understand what pre-training is and have a basic knowledge of the following key technologies:

1. Word2vec
2. ELMo
3. GPT
4. BERT
5. XLNet

What is Pre-Training?

Pre-training in AIrefersto training a model with one task to help it form parameters that can be used in other tasks.

The concept of pre-training is inspired by human beings. Thanks to an innate ability, we don't have to learn everything from scratch. Instead, we transfer and reuse our old knowledge of what we have learned in the past to understand new knowledge and handle a variety of new tasks.

In AI, pre-training imitates the way human beings process new knowledge. That is: using model parameters of tasks that have been learned before to initialize the model parameters of new tasks. In this way, the old knowledge helps new models successfully perform new tasks from old experience instead of from scratch.

5 Essential Pre-Training Technologies

Let's build on our understanding of pre-training by briefly discussing five main methods, each of which will be addressed in further detail in later posts.

1. Word2vec

What is Word2vec? Word2vec is a tool created by Google that produces static word embedding.

In 2013, Google open-sourced a famous tool for word embedding, word2vec, which can be efficiently trained on millions of words. It turns out that the word embedding trained by word2vec has the seemingly magical ability to measure word-to-word similarity.

So let's take a quick look under the hood!

The word2vec uses is a shallow neural network with each word's *one-hot embedding* as its input and output.

What does one-hot embedding look like?

If a dictionary has four words, {'have', 'a', 'good', 'day'}, the one-hot embedding of the word "good" is [0, 0, 1, 0].

One way of training is to predict a target word (the word's one-hot embedding) as output, with the one-hot embedding of its surrounding words as input. Another way is to predict the surrounding words as output with a target word as input, which corresponds individually to CBOW and Skip-gram.

After the training is finished, the parameter matrix of the model will have formed the word-embedding dictionary — so we get each word's embedding of the training data.

In fact, when we talk about the word2vec algorithm or model, it actually refers to the CBOW and Skip-gram models. Many people think that word2vec itself refers to an algorithm or model but this is a misunderstanding.

On a final note, Hierarchical Softmax and Negative Sampling are optimization algorithms for training the CBOW and Skip-gram models.

2. ELMo

What is ELMo? ELMo is an AI technology for dynamic word embedding with LSTM.

To understand ELMo, we must consider that the word embedding learned from Word2Vec is static: no matter the context, it will not change. This leads to a situation where word2vec's word embedding can't handle polysemous words.

But we can adjust the word embedding according to the context to give it a more semantic meaning — which is where ELMo comes into play. It is the idea of dynamically adjusting word embedding according to the current context.

ELMo's network architecture uses a two-layer bidirectional LSTM on top of the word embedding layer learned by word2vec.

To understand how this works, let's work up from the word2vec layer, the product of which is embedding1.

Next, we add two bidirectional LSTM layers, where each layer is composed of two LSTMs — one from left to right and the other one from right to left. This gives us a second and third layer on top of the first. If we take a closer look at layer2, as an example, we'll see that the two LSTMs within it generate two new embeddings.

After concatenating the two embeddings, we have a new *embedding2* of this layer, which we feed into the third layer. The bidirectional LSTM in layer3 generates *embedding3* in the same way as in layer2.

Then all three embeddings are weighted and summed up. Now, a new word embedding is obtained with syntactic (*embedding2,* the second-layer output) and semantic (*embedding3,* the third-layer output) information coded in it.

ELMo's pre-training process not only learns word embedding but also learns a two-layer, bi-directional LSTM network structure — both of which have their own roles when facing new tasks.

3. GPT

What is GPT? In AI, GPT is a transformer-decoder-based autoregressive language model.

The core idea behind the Transformer model is self-attention — the ability to attend to different positions of the input sequence to compute a representation of that sequence. To understand the Transformer, we need to discuss the attention mechanism and self-attention mechanism — which we will in detail in future articles.

You can also refer to Jay Alammar's blog post, The Illustrated Transformer, which gives a very clear explanation of the Transformer. But for now, we only need to understand that the Transformer consists of two parts: an encoder and a decoder.

GPT is a method based on the Transformer *decoder*. GPT also uses static word embedding as input, plus a certain number of layers of the Transformer decoder on top of it.

As an example, let's take another four-word sentence: "w1, w2, w3, w4."

Now let’s take w3 as the current word. After its embedding passes through a decoder layer, it will become a new embedding — which will include the attention information w3 paid to w1 and w2. Note that w1 and w2 are to the left of w3.

We will explain attention information in articles to come but for now, you can simply think of it as a new kind of embedding. This gives the new embedding a better representation to better complete the task of predicting the next word from left to right.

When fine-tuning specific NLP tasks with supervised learning, GPT — unlike ELMo, which connects to other model layers as feature representations — does not need to re-build new model structures for tasks. Instead, it directly connects the last layer to softmax as the task output layer, then fine-tunes the entire model.

4. BERT

What is BERT? In AI, BERT is a Transformer-encoder-based autoencoder language model.

In the paper “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Google divided pre-trained language representation methods into feature-based methods (ELMo) and fine-tuning-based methods (GPT).

BERT is a fine-tuning-based and *encoder*-based method (where you wil recall that GPT is decoder-based).

BERT changes the unidirectional language model in GPT into a bidirectional one. Instead of using the standard left-to-right prediction of the next word as the target task, BERT proposes two new tasks.

BERT’s first pre-training task is called MLM, or Masked Language Model. In the input word sequence of this model, 15% of the words are randomly masked and the task is to predict what they are. What we see is that, unlike previous models, BERT can predict these words from both directions — not just left-to-right or right-to-left.

In addition, in terms of model input, BERT WordPiece embedding is used instead of standard word embedding. That’s because BERT, by default, doesn’t use words as tokens but word pieces. For example, ‘playing’ is ‘play’, ‘ing’ instead of ‘playing’. From here, a position embedding and a segment embedding are added.

The position embedding alleviates the shortcomings of self-attention in ignoring word position information. The segmentation embedding represents the division of two sentences. The two input sentences are represented by 0 and 1 in the segment embedding.

After summing up the WordPiece embedding, position embedding, and segment embedding, we get a new embedding with word information and position information. Then we input this embedding to the BERT Transformer encoder layer to start the training.

5. XLNet

XLNet: XLNet is an AI technology that integrates GPT and BERT

XLNet introduces us to the *autoregressive* model and DAE (denoise *autoencoder)* model.

The language model (LM) that predicts the next possible word based on the *before-context* (words to the left) or predicting the previous word based on the *after-context* (words to the right) is called the autoregressive LM.

Autoregressive language models have naturally matched tasks to generate the next words. The autoencoder model, on the other hand, can naturally integrate into the bidirectional language model. This makes it excellent for tasks that require both before-context and after-context, such as reading comprehension.

The idea behind XLNet is: Can we take advantage of both autoregressive LM and DAE LM?

That is, XLNet tries to introduce the bidirectional language model into the autoregressive language model.

We know that the autoregressive language model reads from left to right, meaning the model can't see the *after-context* — what is to the right of the current word. Knowing how we make the model see words to the right of the current word with an autoregressive LM, which can only see words on the left, is equivalent to know how to implement a bidirectional language model in the autoregressive language model.

XLNet's approach is to fix the current word position in the sentence. That is, if the current word, *w,* is at position 3, then it stays in position 3 but the other words in the sentence are randomly shuffled to different positions. This way, even if the model can only read what is to the right, it still has a way to see the other words in the sentence.

We can get a list of possible sentences, and then randomly choose some sentences among the possibilities as input for model pre-training. In this way, we see both the words on the left and those to the right of the current word, and the form of the autoregressive model remains unchanged.

Of course, in practice, the above ideas are not implemented in the input step but in other processes. But that's a story for another time and something we'll address in a future XLNet article.

In Conclusion

Pre-Training is the most fruitful branch of current NLP research. Arguably, it changed the research way of NLP in the past. Understanding Pre-Training is an essential process for learning AI.

My next article would be:

Attention in NLP: One Key to Making Pre-Training Successful

If you are interested in this article and want to discuss this topic more deeply, please feel free to leave comments or contact me at gglfxsld@gmail.com.