Breaking BERT Down

A complete breakdown of the latest milestone in NLP

Shreya Ghelani

Jul 26, 2019 · 17 min read ★

What is BERT?

BERT is short for Bidirectional Encoder Representations from Transformers. It is a new type of language model developed and released by Google in late 2018. Pre-trained language models like BERT play an important role in many natural language processing tasks, such as Question Answering, Named Entity Recognition, Natural Language Inference, Text Classification etc.

BERT is a multi-layer bidirectional Transformer encoder based on fine-tuning. At this point it is important to introduce the Transformer architecture.

What is a Transformer?

In 2017, Google published a paper titled Attention Is All You Need, which proposed an attention-based structure to deal with sequence model related issues, such as machine translation. Traditional neural machine translation mostly uses RNN or CNN as the model base of encoder-decoder. However, Google's Attention-based Transformer model abandons the traditional RNN and CNN formula. The model works highly in parallel, so training speed is also extremely fast while improving translation performance.

Let's take a step back and understand Attention.

What is Attention?

divided into the encoder layer and the decoder layer, and is composed of RNN or RNN variants (LSTM, GRU, etc.). The encoder vector is the final hidden state produced from the encoder part of the model. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions. It acts as the initial hidden state of the decoder part of the model. The main bottleneck of the Seq2Seq model is the need to compress the entire contents of the source sequence into a fixed-size vector. If the text is slightly longer, it is easy to lose some information of the text. In order to solve this problem, Attention came into being. The attention mechanism alleviates this problem by allowing the decoder to look back at the source sequence hidden state, and then provide its weighted average as an additional input to the decoder. Using Attention, as its name suggests, the model selects the context that best fits the current node as input during the decode phase. There are two main differences between Attention and the traditional Seq2Seq model. First, the encoder provides more data to the decoder, and the encoder will provide the hidden state of all nodes to the decoder, not just the hidden state of the last node of the encoder.

https://jalammar.github.io/images/seq2seq_7.mp4

Second, the decoder does not directly use the hidden state provided by all encoders as input, but adopts a selection mechanism to select the hidden state that best matches the current position. To do so, it tries to determine which hidden state is most closely related to the current node by calculating the score value of each hidden state and doing a softmax calculation over the scores, which allows the higher correlation of the hidden state to have a larger fractional value, and the less relevant hidden state has a lower fractional value. It then multiples each hidden state by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores. This scoring exercise is done at each time step on the decoder side.

https://jalammar.github.io/images/attention_process.mp4

Let us now bring the whole thing together in the following visualization and look at how the attention process works:

1. The attention decoder RNN takes in the embedding of the <END> token, and an

3. Attention Step: We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network (one trained jointly with the model).
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps

https://jalammar.github.io/images/attention_tensor_dance.mp4

Coming back to Transformers

Transformer model uses the encoder-decoder architecture. In the paper published by Google, the encoder layer is stacked by 6 encoders, and the decoder layer is the same. The internal structure of each encoder and decoder is as follows -

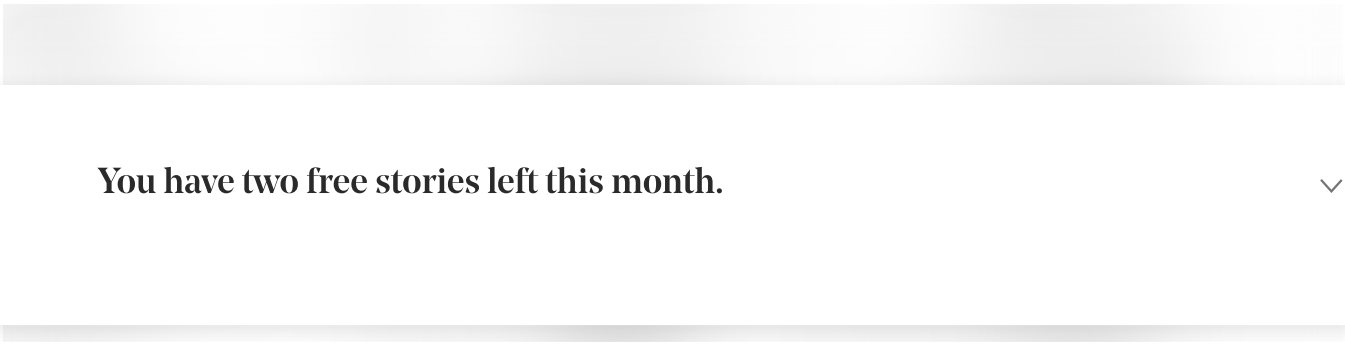
The Encoder consists of two layers, a self-attention layer and a feedforward neural network. The self-attention helps the current node not only focus on the current word, but also obtain the semantics of the context. The Decoder also contains the two-layer network mentioned by the encoder, but there is also an attention layer in the middle of the two layers to help the current node get the key content that needs attention.

Let’s break down the individual components.

Self-Attention

respectively. These three vectors are produced by multiplying the word embedding vectors and a randomly initialized matrix (the dimension is (64, 512) in the paper)whose value is updated during the back-propagation process.

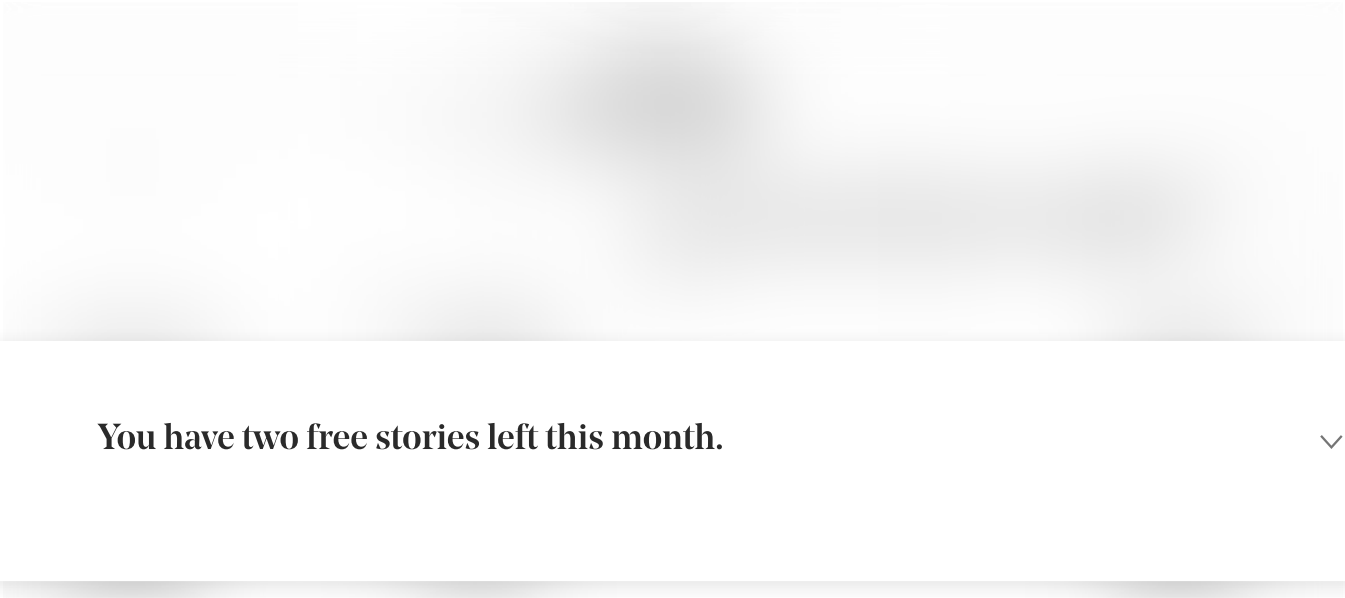
Next, we calculate the fractional value of the self-attention, which determines how much attention is paid to the rest of the input sentence when we encode a word at a certain location. The calculation method of this fractional value uses the Query and Key vectors. Then we divide the result by a constant. Here we divide by 8. This value is generally the square root of the first dimension of the matrix mentioned above, that is, the square root 8 of 64. Then we run a softmax calculation over all the scores. The result is the relevance of each word to the word at the current position. Naturally, the word relevance of the current position will definitely be large. The last step is to multiply the Value vector with the softmax results and add them. The result is the value of self-attention at the current node.



This method of determining the weight distribution of value by the degree of similarity between query and key is called scaled dot-product attention.

Multi-Headed Attention

The more powerful part of this paper is the addition of another mechanism to self-attention, called “multi-headed” attention, which doesn’t just initializes a set of Q, K, V matrices. Instead, multiple groups are initialized, and transformer uses 8 groups, so the final result is 8 matrices.



The feedforward neural network can't accept 8 matrices so we need a way to reduce 8 matrices to 1. To do that, we first connect 8 matrices together to get a large matrix, then multiply this combined matrix with a randomly initialized matrix to get a final matrix. Let's look at the full process.

The Transformer uses multi-head attention in three different ways:

1. In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.

2. The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.

(setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. This will be explored in more detail in the Decoder section where we talk about Masking.

Positional Encoding

So far, we don't have a way of interpreting the order of words in an input sequence in the transformer model. To handle this problem, the transformer adds an additional vector Positional Encoding to the input of the encoder and decoder layers. The dimension is the same as the embedding dimension. The value of this Positional Encoding is added to the value of embedding and sent to the next layer as input. There are many options for location coding, both learned and fixed.

Residual Connection and Layer Normalization

In both the Encoder and Decoder, a residual connection is employed around each of the two sub-layers, followed by layer normalization. Skip connections or residual connections are used to allow gradients to flow through a network directly, without passing through non-linear activation functions. Non-linear activation functions, by nature of being non-linear, cause the gradients to explode or vanish (depending on the

layer 2 to layer 3. This happens because, as the network learns and the weights are updated, the distribution of outputs of a specific layer in the network changes. This forces the higher layers to adapt to that drift, which slows down learning. After normalizing the input in the neural network, we don't have to worry about the scale of input features being extremely different. To understand layer normalization, it is useful to contrast it with batch normalization. A mini-batch consists of multiple examples with the same number of features. Mini-batches are matrices — or tensors if each input is multi-dimensional — where one axis corresponds to the batch and the other axis — or axes — correspond to the feature dimensions. Batch normalization normalizes the input features across the batch dimension. The key feature of layer normalization is that it normalizes the inputs across the features. In batch normalization, the statistics are computed across the batch and are the same for each example in the batch. In contrast, in layer normalization, the statistics are computed across each feature and are independent of other examples.

Bringing both the residual connection and layer normalization together.

Decoder

Going back to the Transformer architecture diagram, we can see that the Decoder part is similar to the encoder part, but there is a masked multi-head attention at the bottom. Mask represents a mask that masks certain values so that they do not have an effect when the parameters are updated. There are two kinds of masks in the Transformer model — padding mask and sequence mask. The padding mask is used in all the scaled dot-product attention, and the sequence mask is only used in the decoder's self-attention.

A padding mask solves the problem of input sequences being of variable length. Specifically, we pad 0 after a shorter sequence. But if the input sequence is too long, the content on the left is intercepted and the excess is discarded directly. Because the location of these fills is actually meaningless, our attention mechanism should not focus on these locations, so we need to do some processing. The specific approach is to add a very large negative number (negative infinity) to the values of these positions, so

A sequence mask is designed to ensure that the decoder is unable to see future information. That is, for a sequence, at time_step t, our decoded output should only depend on the output before t, not the output after t. This is specific to the Transformer architecture because we do not have RNNs where we can input our sequence sequentially. Here, we input everything together and if there were no mask, the multi-head attention would consider the whole decoder input sequence at each position. We achieve this by generating an upper triangular matrix with the values of the upper triangles all zero and applying this matrix to each sequence.

For the self-attention of the decoder, the scaled dot-product attention is used, and the padding mask and sequence mask are added as the attn_mask. In other cases, attn_mask is equal to padding mask.

Another detail is that the the decoder input will be shifted to the right by one position. One reason to do this is that we do not want our model to learn how to copy our decoder input during training, but we want to learn that given the encoder sequence and a particular decoder sequence, which has been already seen by the model, we predict the next word/character. If we don't shift the decoder sequence, the model learns to simply 'copy' the decoder input, since the target word/character for position *i* would be the word/character *i* in the decoder input. Thus, by shifting the decoder input by one position, our model needs to predict the target word/character for position *i* having only seen the word/characters *1, …, i-1* in the decoder sequence. This prevents our model from learning the copy/paste task. We fill the first position of the decoder input with a start-of-sentence token, since that place would otherwise be empty because of the right-shift. Similarly, we append an end-of-sentence token to the decoder input sequence to mark the end of that sequence and it is also appended to the target output sentence.

Output layer

After the decoder layer is fully executed, in order to map the resulting vector to words from a vocabulary, a fully connected layer and softmax layer is added at the end.

The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits

layer.The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

Coming back to BERT

BERT is based on the Transformer architecture. It is a deep, two-way deep neural network model. BERT's key technical innovation is applying the bidirectional training of Transformer to language modeling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. BERT employs a novel technique named Masked Language Modeling (as we will see later) which allows bidirectional training in models which was previously impossible. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary.

BERTBASE: L=12, H=768, A=12, Total Parameters=110M

BERTLARGE: L=24, H=1024, A=16, Total Parameters=340M

There are two phases to using BERT: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. A distinctive feature of BERT is its unified architecture across different tasks. There is minimal difference between the pre-trained architecture and the final downstream architecture. During fine-tuning, all parameters are fine-tuned.

BERT Pre-Training Process

The BERT pre-training phase consists of two unsupervised predictive tasks, one is the Masked Language Model and the other is Next Sentence Prediction.

Masked Language Model — Because of the two-way function (bi-directionality) and the effect of the multi-layer self-attention mechanism that BERT uses, in order to train a deep bidirectional representation, some percentage (15% in the paper) of the input tokens are simply masked at random, and then those masked tokens are predicted. The

Although this allows to obtain a bidirectional pre-trained model, a downside is that there is a mismatch between pre-training and fine-tuning, since the [MASK] token does not appear during fine-tuning. To mitigate this, the authors do not always replace "masked" words with the actual [MASK] token. The training data generator chooses 15% of the token positions at random for prediction. If the i-th token is chosen, it is replaced with (1) the [MASK] token 80% of the time (2) a random token 10% of the time (3) the unchanged i-th token 10% of the time. The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness.

Next Sentence Prediction —. In order to train a model that understands sentence relationships along with semantic relationships between words, BERT also pre-trains for a binarized next sentence prediction task that can be very easily generated from any text corpus. Select some sentences for A and B, where 50% of the data B is the next sentence of A, and the remaining 50% of the data B are randomly selected in the corpus, and learn the correlation. 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-trained model can better adapt to such tasks.

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

To predict if the second sentence is indeed connected to the first, the following steps

2. The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).

3. Calculating the probability of IsNextSequence with softmax.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

Tokenization — BERT doesn't look at words as tokens. Rather, it looks at WordPieces. It means that a word can be broken down into more than one sub-words. This kind of tokenization is beneficial when dealing with out of vocabulary words, and it may help better represent complicated words.

BERT Model Input

The input of BERT can be a single sentence or a sentence pair in a sequence of words (for example, [question, answer]). For a given word, its input representation can be composed of three-part Embedding summation. The visual representation of Embedding is shown below:

Token Embeddings represents the word vector. The first word is the CLS flag, which can be used for subsequent classification tasks. For non-classification tasks, the CLS flag can be ignored. Segment Embeddings is used to distinguish between two sentences, because pre-training is not only a language model but also a classification task with two sentences as input. Position Embeddings encode word order.

BERT Fine-Tuning for Downstream NLP Tasks

etc. At the output, the token representations are fed into an output layer for token level tasks, such as sequence tagging or question answering, and the [CLS] representation is fed into an output layer for classification, such as entailment or sentiment analysis. Compared to pre-training, fine-tuning is relatively inexpensive.

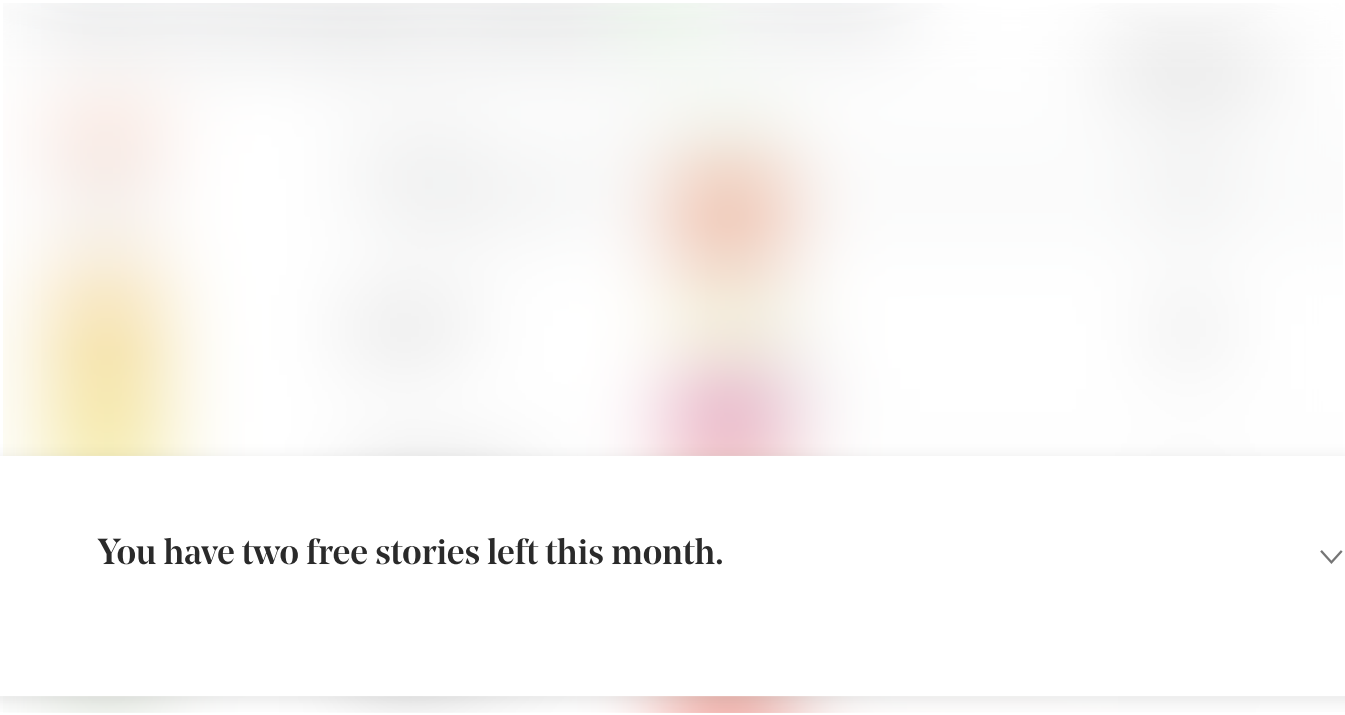
BERT can be used for a wide variety of language tasks, while only adding a small layer to the core model:

1. Classification tasks such as sentiment analysis are done similarly to Next Sentence classification, by adding a classification layer on top of the Transformer output for the [CLS] token.
2. In Question Answering tasks (e.g. SQuAD v1.1), the software receives a question regarding a text sequence and is required to mark the answer in the sequence. Using BERT, a Q&A model can be trained by learning two extra vectors that mark the beginning and the end of the answer.

BERT for feature extraction

The fine-tuning approach isn't the only way to use BERT. You can use the pre-trained BERT to create contextualized word embeddings. Then you can feed these embeddings to your existing model — a process the paper shows yield results not far behind fine-tuning BERT on a task such as named-entity recognition.

Which vector works best as a contextualized embedding? It depends on the task. The paper examines six choices (Compared to the fine-tuned model which achieved a score of 96.4):



Hope you enjoyed reading this post as much as I enjoyed writing it! I would like to acknowledge and thank some really amazing resources on the web (listed in References) that helped me distill the concepts behind BERT and that I have borrowed from extensively while compiling this post.

References

Attention Is All You Need

Ashish Vaswani* Google Brain
Noam Shazeer* Google Brain
Niki Parmar* Google Research
Jakob Uszkoreit* Google Research
Llion Jones* Google Research
Aidan N. Gomez* † University of Toronto
Łukasz Kaiser* Google Brain
Illia Polosukhin* ‡

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [and gated recurrent [neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and initial codebase, and various parts of and assively accelerating

Page 1 / 15

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models , BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

1 Introduction

Language model pre-training has been shown to be effective for improving many natural language processing tasks

There are two existing strategies for appl ing pre-trained language representations to dow stream tasks: *feature-based* and *fine-tuning*. T feature-based approach, such as ELMo , uses task-specific architectures tl include the pre-trained representations as ad tional features. The fine-tuning approach, such the Generative Pre-trained Transformer (Open GPT) , introduces minin task-specific parameters, and is trained on t downstream tasks by simply fine-tuning *all* p trained parameters. The two approaches share t same objective function during pre-training, wh they use unidirectional language models to le general language representations.

We argue that current techniques restrict t power of the pre-trained representations, esp cially for the fine-tuning approaches. The m jor limitation is that standard language models unidirectional, and this limits the choice of arc tectures that can be used during pre-training. F example, in OpenAI GPT, the authors use a left- right architecture, where every token can only tend to previous tokens in the self-attention lay of the Transformer . Such strictions are sub-optimal for sentence-level tas and could be very harmful when applying fi tuning based approaches to token-level tasks su as question answering, where it is crucial to inc porate context from both directions.

ove the fine-tuning bas g BERT: **B**idirectio s from **T**ransforme

natural

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)

Translations: Chinese (Simplified), Korean Watch: MIT's Deep Learning State of the Art lecture referencing this post...

jalammar.github.io

The Illustrated Transformer

Discussions: Hacker News (65 points, 4 comments), Reddit r/MachineLearning (29 points, 3 comments) Translations...

jalammar.github.io

Machine Learning Deep Learning Naturallanguageprocessing Transfer Learning

Data Science

About Help Legal