# Understanding BERT Part 2: BERT Specifics

Francisco Ingham Follow
Nov 26, 2018 · 12 min read


Discover the wonderful world of bidirectionality

*This is Part 2/2 of Understanding BERT written by Miguel Romero and Francisco Ingham. Each article was written jointly by both authors. If you did not already, please refer to Part 1 to understand the Encoder architecture in depth before reading this article.*

*Many thanks to Yannet Interian for her revision and feedback.*

In Part 2 we are going to dive deeper into what makes BERT specially effective in varied NLP tasks, achieving, in the authors' own words:

> *[…] new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE benchmark to 80.4% (7.6% absolute improvement), MultiNLI accuracy to 86.7% (5.6% absolute improvement) and the SQuAD v1.1 question answering Test F1 to 93.2 (1.5 absolute improvement), outperforming human performance by 2.0.*

To avoid confusions, let's clearly distinguish between papers and architectures before we begin. We are going to be referencing three papers with three different (albeit related) architectures:

1. **'Transformer'** will be our name for the architecture used in the paper that introduced the transformer, Attention Is All You Need.
2. **'OpenAI GPT'** will reference OpenAI's work with the transformer architecture which they pretrained on a language model and later tested in a number of downstream tasks.
3. **'BERT'** will reference Google's paper on which they modified OpenAI GPT's architecture to outperform its predecessor in most NLP tasks.

# Introduction

*What is BERT?*

**BERT** stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. It is basically a bunch of Transformer encoders stacked together (not the whole **Transformer** architecture but just the encoder). The concept of bidirectionality is the key differentiator between **BERT** and its predecessor, **OpenAI GPT**. **BERT** is bidirectional because its self-attention layer performs self-attention on both directions. Let's see an example.

Let's say our input sentence is 'I love to work on NLP'. In **OpenAI GPT**, the 'love' token would only have a self-attention relationship with the 'I'

token and with itself (only backwards). In **BERT** the same token would have self-attention with *every other token in the sentence*. But why didn't **OpenAI GPT** use bidirectionality if the original encoder in **Transformer** uses bidirectionality? Well, this is explained by the tasks they were trained on.

## Architectures and bidirectionality

*Why didn't OpenAI pre-train their model with bidirectionality?*

**OpenAI GPT** pre-trained its model on a language model task. This means that they were training the transformer encoder to predict, for each token, what would be the next word in the sentence. If they were to perform bidirectional self-attention, then the model would learn that the next word in the sentence is the target and would predict it always, with 100% accuracy. In our example, to predict the next word after 'love', it would just need to peek into the next word 'to' and select it as a prediction (it would soon learn that the next word has 100% correlation with the target). In this way, the model would not be learning anything useful neither about the grammar or syntax of the language nor the meanings of the tokens and would not be useful for downstream tasks.

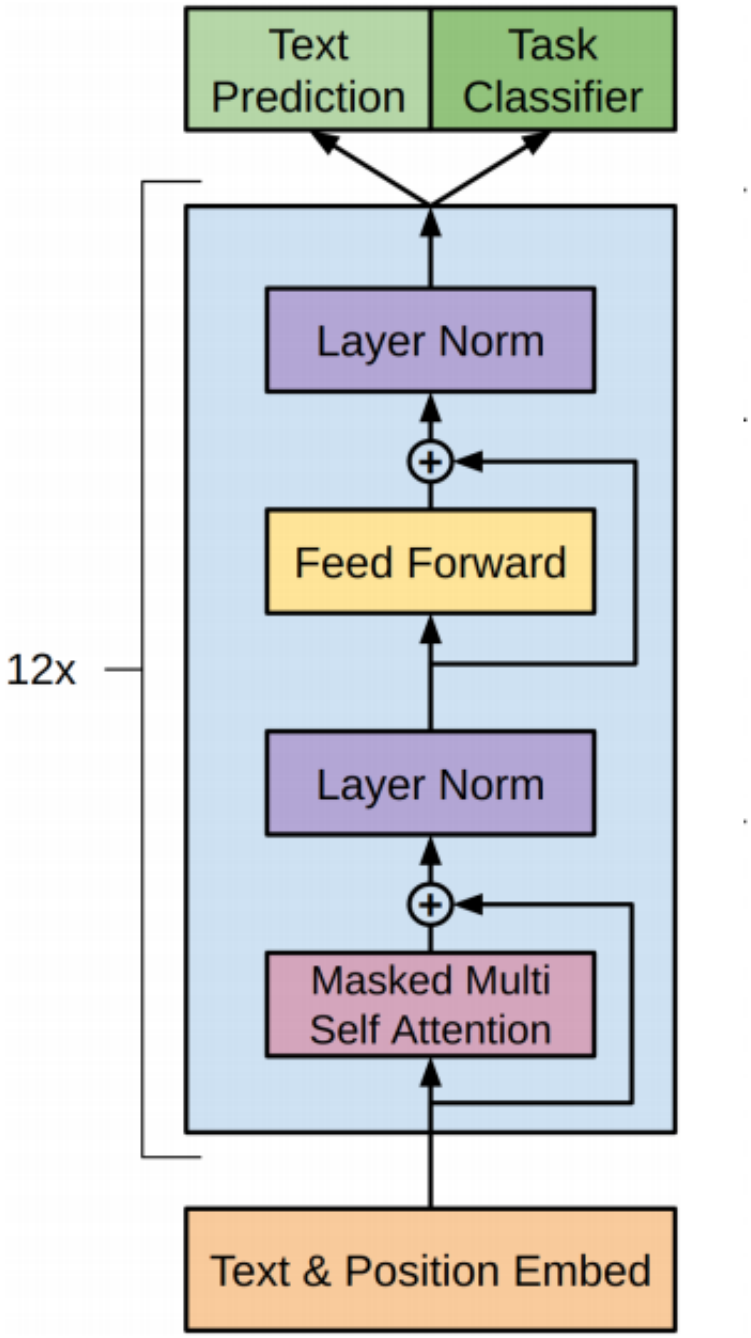


**Figure 1: OpenAI GPT** masks the input when performing self-attention

*How did the **Transformer** handle bidirectionality?*

To start with, it was trained by solving another problem, namely translating English to German. It is also important to consider that the **Transformer** architecture consists of an Encoder and a Decoder and the Encoder's output is an input to the Decoder. So there is no problem in

the **Transformer** performing self-attention between all the Encoder input tokens since none of these were part of the prediction (the prediction was the translated German sequence and this *was* masked, for more information see our article on the Decoder).

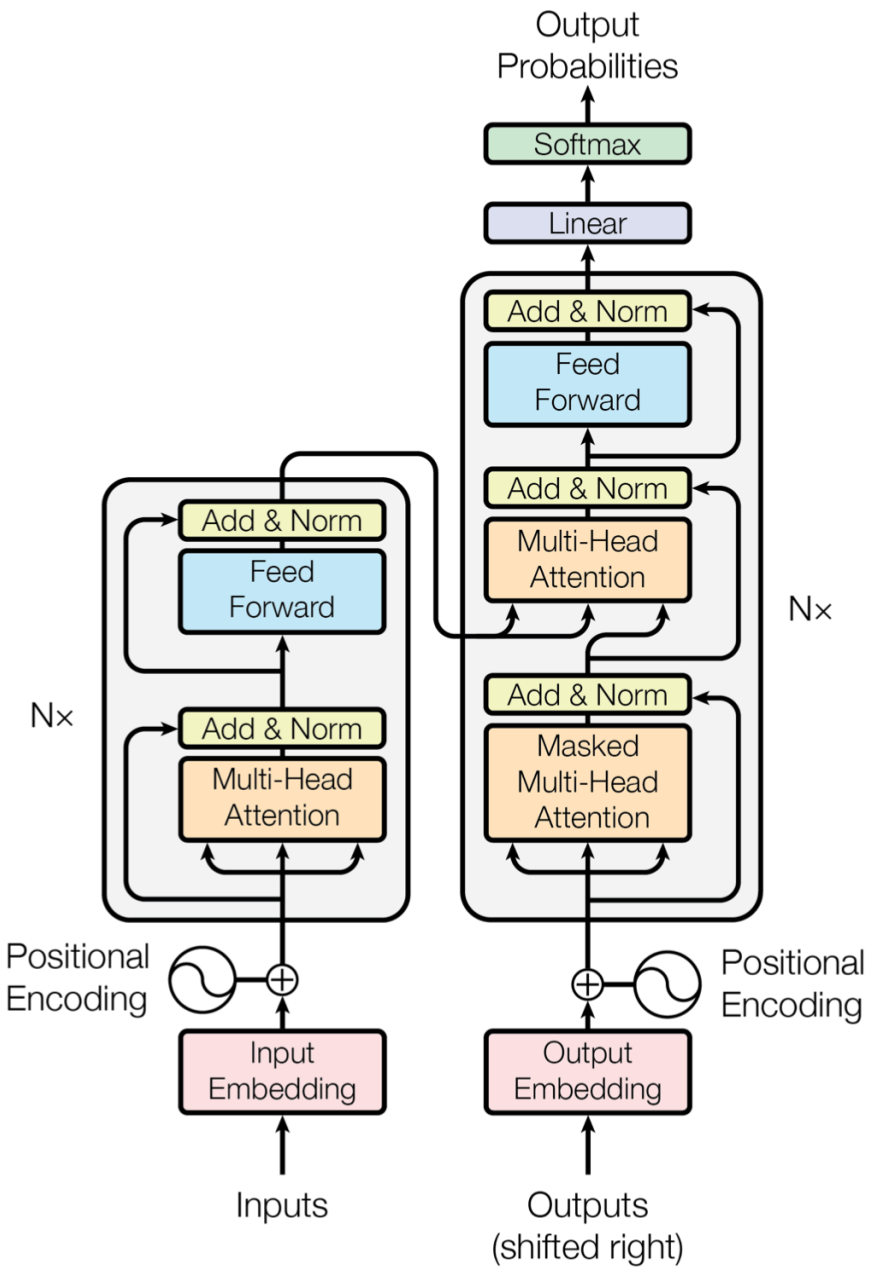


Figure 2: The **Transformer** Architecture

*And how did **BERT** handle bidirectionality?*

We start getting into the crux of what makes **BERT** special. Remember that adding bidirectionality while using the Transformer Encoder to train a language model is unfeasible (this is **OpenAI GPT)**. We know **BERT** did use bidirectionality and kept the original architecture intact so that only leaves one suspect… you guessed right, they changed the problem! Instead of pre-training the model on a language model, they pre-trained it into the "masked language model" and "next sentence prediction". Before we dive into these problems and how they differ from **OpenAI GPT** let's spend a few minutes discussing BERT's input embeddings.

## Input embeddings

BERT presents a few differences with the **Transformer** in how they handle input sequences with the objective of increasing the number of tasks the model can work on.

As you can see in Figure 3, the components of the **Transformer** are still there: token embeddings and positional embeddings. However, this time they used *pretrained* token embeddings (WordPiece) and *learned* positional embeddings (which, as specified in Attention is All You Need, have similar performance to the positional embedding functions used in that paper). There are also some new tricks introduced by the authors:

- To be able to solve two sentence problems like question answering and next sentence prediction (Pre-training Task 2), they added a [SEP] token to mark the end of a sentence and added a sentence embedding (this embedding is constant for each sentence but different across two sentences). This allows the model to easily determine where one sentence ends and the other begins and to give different meanings to different sentences.
- To allow for classification tasks, the authors added a [CLS] token at the beginning of the input sequence, to be used only if classifying.

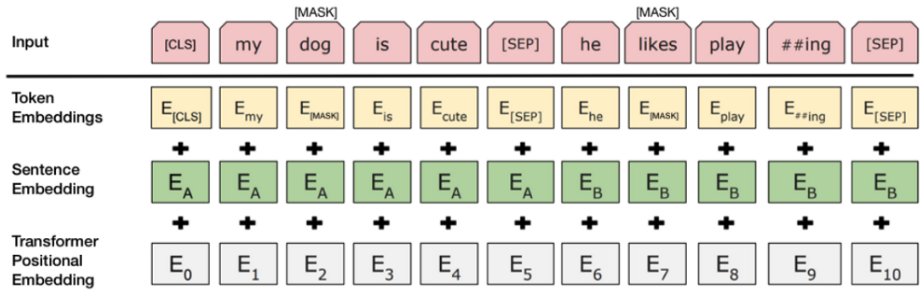


Figure 3: Embeddings

# Pre-training

The model was trained in two tasks simultaneously: *Masked Language Model* and *Next Sentence Prediction*.

**Task 1: Masked Language Model**

The *Masked Language Model* asks the model to predict, not the next word for a sequence of words, but rather random words from within the sequence.

*How were tokens masked?*

In this specific case, 15% of the words that were fed in as input were masked. But not all tokens were masked in the same way. To explain how the tokens were masked, we will use the same example that the authors used in the paper: 'My dog is hairy'. Let's assume that this appearance of 'hairy' was among the tokens chosen to be replaced.

The tokens were replaced in the following way:

- 80% were replaced by the '<MASK>' token

> *Example: "My dog is* ***<MASK>****"*

- 10% were replaced by a random token

> *Example: "My dog is* **apple***"*

- 10% were left intact

> *Example: "My dog is* **hairy***"*

*Why did they not use a '<MASK>' replacement token all around?*

If the model had been trained on only predicting '<MASK>' tokens and then never saw this token during fine-tuning, it would have thought that there was no need to predict anything and this would have hampered performance. Furthermore, the model would have only learned a contextual representation of the '<MASK>' token and this would have made it learn slowly (since only 15% of the input tokens are masked). By sometimes asking it to predict a word in a position that did not have a '<MASK>' token, the model needed to learn a contextual representation of *all* the words in the input sentence, just in case it was asked to predict them afterwards.

*Are not random tokens enough? Why did they leave some sentences intact?*

Well, ideally we want the model's representation of the masked token to be better than random. By sometimes keeping the sentence intact (while still asking the model to predict the chosen token) the authors biased the model to learn a meaningful representation of the masked tokens.

*Will random tokens confuse the model?*

The model will indeed try to use the embedding of the random token to help in its prediction and it will learn that it was actually not useful once it sees the target (correct token). However, the random replacement happened in 1.5% of the tokens (10%*15%) and the authors claim that it did not affect the model's performance.

*The model will only predict 15% of the tokens but language models predict 100% of tokens, does this mean that the model needs more iterations to achieve the same loss?*

Yes, the model does converge more slowly but the increased steps in converging are justified by an considerable improvement in downstream performance.

## Task 2: Next Sentence Prediction

This task consists on giving the model two sentences and asking it to predict if the second sentence follows the first in a corpus or not.

*Why is a second task necessary at all?*

The authors pre-trained their model in *Next Sentence Prediction* because they thought important that the model knew how to relate two different sentences to perform downstream tasks like question answering or natural language inference and the "masked language model" did not capture this knowledge. They prove that pre-training with this second task notably increases performance in both question answering and natural language inference.

*What percentage of sentences where actually next sentences?*

50% of the sentences were paired with actual adjacent sentences in the corpus and 50% of them were paired with sentences picked randomly from the corpus.

## Pre-training procedure

The pre-training corpus was built from BookCorpus (800M words) and English Wikipedia (2,500M words). Tokens were tokenized using 37,000 WordPiece tokens.

To generate the pre-training sequences, the authors got random samples in batches of two (50% of the time adjacent to each other) such that the combined length of the two chosen sentences was $\leq 512$ tokens. Once each sequence was built, 15% of its tokens were masked.

An example of a pre-training sequence presented in the paper is:

> *Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]*

In this case the sentences are adjacent, so the label in [CLS] would be '<IsNext>' as in:

> *Input = <IsNext> the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]*

The loss was calculated as the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood.

# Fine-tuning

## Fine-tuning procedure

The fine-tuning is straightforward since the model as described previously can handle a wide array of downstream tasks. The only difference with the pre-training architecture is related to multi-label classification.

Since the '[CLS]' token can only output one number and a multi-label classification problem needs to have an output dimension equal to the number of labels, the authors needed to transform the output to match the required dimension. They did this by adding a linear layer with a softmax activation function to the last hidden state for the first token (which corresponds to the vector in the '[CLS]' token position). If we call the hidden vector for our last token *C* with dimension *h* and our weight matrix *W* with dimensions *(k,h)* where *k* is the number of labels, then:

$$softmax(CW^t) \in \mathbb{R}^k$$

which has exactly the dimensions we need to perform multi-label classification.

# Testing and Results

BERT has achieved SOTA in 11 varied NLP tasks. Before we dive into results we should clarify an important aspect of the chosen architecture.

The paper presents two versions of BERT: **BERT base** and **BERT large**. **BERT base** was explicitly created to compare results with **OpenAI GPT** and control for model size*. As such, **BERT base** has exactly the same parameters as **OpenAI GPT**: L=12, H=768, A=12 where L is the number of stacked encoders, H is the hidden size and A is the number of heads in the MultiHead Attention layers. **BERT large** is basically larger and more compute-intensive: L=24, H=1024, A=16.

*Why BERT?*

**BERT** is amazing. With some modifications to **OpenAI GPT**, it shattered SOTA for many of the most important NLP tasks.

The two clear conclusions we can derive from the paper's results are that **BERT base** beat **OpenAI GPT** and that **BERT large** beat **BERT base**. These results suggest that the architectural modifications the authors made on **OpenAI GPT** (mainly bidirectionality) and the increase in parameters from **BERT base** to **BERT large** played a role in considerably improving the model's natural language understanding. The authors confirmed these results with ablation studies (see *Ablation Studies*).

With these conclusions in mind, let's investigate the results in detail.

## GLUE

> *The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) is a collection of diverse natural language understanding tasks.*

8 GLUE classification tasks were used to assess performance. As we can see in Figure 5, **BERT base** not only beat **OpenAI GPT** on all tasks achieving SOTA, but it improved SOTA by an impressive 5% on average. **BERT large** beat **BERT base** for all tasks as well.

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

Figure 4: **BERT** achieved SOTA on all GLUE-tested tasks

## SQUAD

> *The Standford Question Answering Dataset (SQuAD) is a collection of 100k crowdsourced question/answer pairs (Rajpurkar et al., 2016). Given a question and a paragraph from Wikipedia containing the answer, the task is to predict the answer text span in the paragraph.*

In SQUAD the big improvement in performance was achieved by **BERT large**. The model that achieved the highest score was an ensemble of **BERT large** models, augmenting the dataset with TriviaQA.

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

Figure 5: **BERT** achieved SOTA in SQUAD

## Named Entity Recognition

> *To evaluate performance on a token tagging task, we fine-tune BERT on the CoNLL 2003 Named Entity Recognition (NER) dataset. This dataset consists of 200k training words which have been annotated as Person, Organization, Location, Miscellaneous, or Other (non-named entity).*

In NER, **BERT large** achieved SOTA but did not push the bar too high.

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

Figure 6: **BERT** achieved SOTA in NER

**SWAG**

> *The Situations With Adversarial Generations (SWAG) dataset contains 113k sentence-pair completion examples that evaluate grounded commonsense inference (Zellers et al., 2018).*

In this task, **BERT large** improved SOTA by a staggering 27.1% which is superhuman performance!

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

Figure 7: **BERT** achieved SOTA in SWAG

## Ablation Studies

The authors also performed numerous useful ablation studies. Although we are not going to deep dive into each of them, we are going to mention the most important ones. Their findings were:

- *Most of the difference in performance between **BERT** and **OpenAI GPT** can be explained by bidirectionality.*

In most of the tasks bidirectionality was responsible for a much greater part of the increase in performance than the other big change, namely using next sentence prediction. The exception of course was QNLI, a binary Question Answering task (since Next Sentence Prediction is a close problem to Question Answering than Masked Language Models).

| Tasks | Dev Set | | | | |
|---|---|---|---|---|---|
| | MNLI-m (Acc) | QNLI (Acc) | MRPC (Acc) | SST-2 (Acc) | SQuAD (F1) |
| $\text{BERT}_{\text{BASE}}$ | 84.4 | 88.4 | 86.7 | 92.7 | 88.5 |
| No NSP | 83.9 | 84.9 | 86.5 | 92.6 | 87.9 |
| LTR & No NSP | 82.1 | 84.3 | 77.5 | 92.1 | 77.8 |
| + BiLSTM | 82.1 | 84.1 | 75.7 | 91.6 | 84.9 |

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

Figure 8: Comparison of different **BERT** variants.

- *Bigger is better. Much bigger is much better. Not only for large scale but also for very small scale tasks.*

Using **BERT large** improved performance from **BERT base** in GLUE selected tasks even if **BERT base** already had a great number of parameters (110M) compared to the largest tested model in **Transformer** (100M).

| Hyperparams | | | | Dev Set Accuracy | | |
|---|---|---|---|---|---|---|
| #L | #H | #A | LM (ppl) | MNLI-m | MRPC | SST-2 |
| 3 | 768 | 12 | 5.84 | 77.9 | 79.8 | 88.4 |
| 6 | 768 | 3 | 5.24 | 80.6 | 82.2 | 90.7 |
| 6 | 768 | 12 | 4.68 | 81.9 | 84.8 | 91.3 |
| 12 | 768 | 12 | 3.99 | 84.4 | 86.7 | 92.9 |
| 12 | 1024 | 16 | 3.54 | 85.7 | 86.9 | 93.3 |
| 24 | 1024 | 16 | 3.23 | 86.6 | 87.8 | 93.7 |

Figure 9: Different **BERT** sizes' performance on select downstream GLUE tasks.

*If you have any feedback please let us know in the comment section!*

*There are other differences between OpenAI GPT and BERT which we will quote directly from the paper:

> *GPT is trained on the BooksCorpus (800M words); BERT is trained on the BooksCorpus (800M words) and Wikipedia (2,500M words).*

> *GPT uses a sentence separator ([SEP]) and classifier token ([CLS]) which are only introduced at fine-tuning time; BERT learns [SEP], [CLS] and sentence A/B embeddings during pre-training.*

> *GPT was trained for 1M steps with a batch size of 32,000 words; BERT was trained for 1M steps with a batch size of 128,000 words.*

> *GPT used the same learning rate of 5e-5 for all fine-tuning experiments; BERT chooses a task-specific fine-tuning learning rate which performs the best on the development set.*

## References

Cover image source: http://muppet.wikia.com/wiki/Bert%27s_books

*Attention Is All You Need.* Vaswani et al, Dec 2017.

*Improving Language Understanding by Generative Pre-Training.* Radford et al, 2018. OpenAI.

*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* Jacob Devlin et al., Google AI Language. Oct 2018.