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

Abdullah Aml

June 5, 2022

Abstract

We present a new model Bidirectional Encoder Representation form Transformers (BERT. The title seems hard and messy, but we will show INSHAA ALLAH that is really a simple idea with a really big results and applications. BERT is an unsupervised pre-training model trained can be used in many applications with just small fine-tuning (IE: adding some linear layers) and achieved state of the art results in 11 Natural Language Processing (NLP) 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 answer- ing 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 Background

In order to understand the paper fully we will introduce some necessary concepts.

1.1 Masked Language Modeling

In order to model a language is to predict the missing word in a sentence. one of these models is Masked Language Model (MLM) we mask a word (cover it) and train our model in order to predict correctly (i.e: model language). Old techniques involves Unidirectional Recurrent Neural Networks (RNNs) as in Figure 1 we have input word "The sky is blue" we masked the verb "is".

The input to our model is the embedding of the words (contracted vector representation that preserve meaning see this guide). h_i represents the contextual meaning of from beginning of sentence(The $-> h_1$) to the current word (h_i) in a Left to Right (LTR) approach. the problem with this in our example we only make predictions of (h_3) captures context from left only, so the model will not differentiate between"is", and "goes" : "The sky (is) blue", "The sky (goes) blue", we have not carpeted the right context "blue" which is adjective.

The solution for this is to add another RNN the captures context form Right to Left (RTL) represnted in Figure 2 by ($\hat{h}_i$). we then concatinate (h_i and $\hat{h}_i$) to apply a liner layer and softmax to get our masked word. In our example will succesfully predict word "is".

But again we have a problem: if the sentence is very big like: "The ball that children plays with in school is blue". If we mask word "is": "The ball that children plays with in school – blue". Will the model give the verb to "ball (i.e: masked word will be "is"), or "children" (i.e: masked word will be "are") ?

To solve this problem we introduce Attention Figure 3. What if we multiply every $[h_i; \hat{h}_i]$ by a factor α_i represents the relevance the masked word with every $word_i$ (i.e: $token_i$) in the sequence. α is calculated with small Feed Forward Network (FFN), or Dot product between the masked word $[h_3; \hat{h}_3]$ and $[h_{masked}; \hat{h}_{masked}]$ as follows in Figure 3:

$$\alpha_i = h_{masked} \cdot h_i$$

$$d_{masked} = \sum_{i}^{N} \alpha_i h_i$$

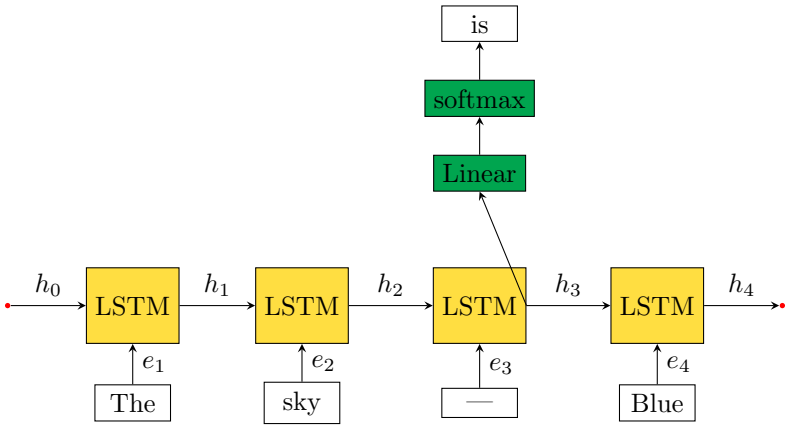


Figure 1: Unidirectional Masked Language Model

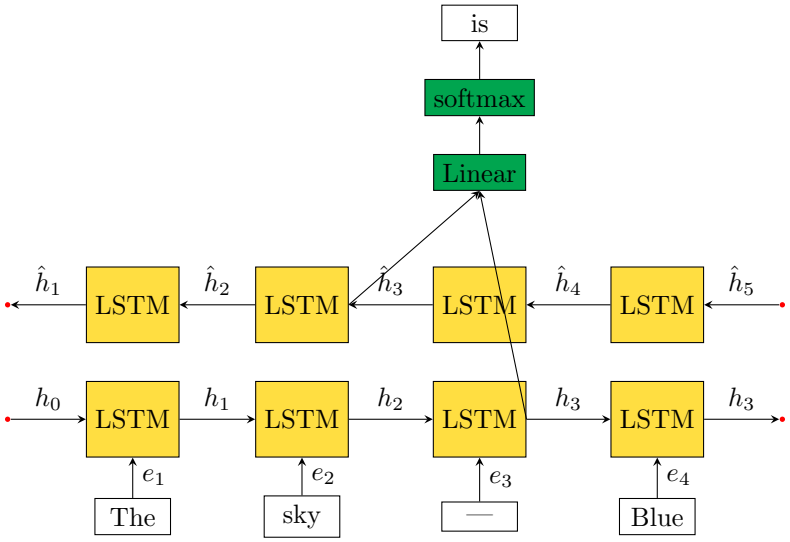


Figure 2: Bidirectional Masked Language Model

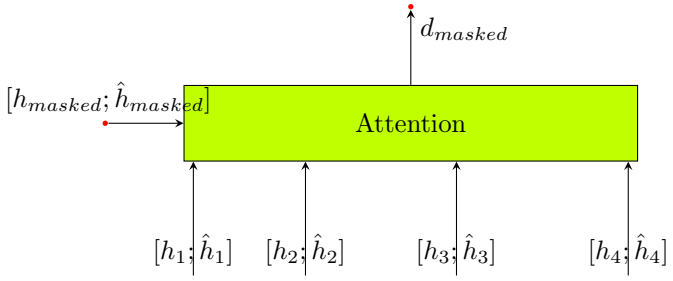


Figure 3: Attention block diagram

This way as in Figure 4 we have solved problems with long sequences.
But bidirectional RNNs are very costly at taring because of their sequential nature. what if we replace RNNs with attention Figure 5 we will use attention for every word (token) in the sentence to get d_i (full context representation) as in Figure 5, we introduce self-attention aka: Transformers.

1.2 Transformers

We will explain transformers in abstract way to learn about transformers in details I recommend these resource at the following order:

- Videos by Eng Ahmed Fathi very very powerful. We build the background over it.
- This blog post by Jay.
- The paper itself: Attention is all you need.
- Step by step explained Implementation Notebook using Pytorch.

We start form the last section as we introduced the "Self Attention" replacing RNNs. We said the new contextual representation will be calculated using self attention for every token (i.e: word) as in Figure 6, but this way Will we have different "self Attention" block for every token ? the answer is No. the weights of the self attention is shared among all "Self Attention" blocks in a way ilustrated in the previous section, or can be tackled deeper from the above resources.

Then we can stack (A) multiple parallel different "self Attention" blocks aka "heads" with different parameters to form the Encoder block of the transformer is in Figure 7. Then we stack (L) Encoder to form our full language model as in Figure 8

2 Introduction

Pre-training for language modeling (LM) has shown its effectiveness, as we need to make accurate models for some hard tasks like Language Inferencing, and Question Answer (QA) problems. The previous pre-trained models has two ways to be applied for downstream tasks:

- Feature Based: uing the pre-trained model as a feature extractor, then applying our architecture for our downstream task. The state of the art of using this techinque is ELMo [SWWP+21] which uses bidirectional LSTM.
- Fine Tuning: fine-tune the pre-trained model with few modifications for the downstream task. The state of the art was Open Ai GPT [RNSS18]. the uses LFT Encoder of the Transformer.

We argue that we have to use bidirectional LM in order to fully capture the full context of the sentences, so we Introudce BERT: Bidirectional Encoder Representation from Transformers. BERT differs previous approaches by:

- Using Masked Language Model (MLM) explained at the background which masks (i.e: hides) 15% of the words to make BERT the hidden words form the input Sequence.

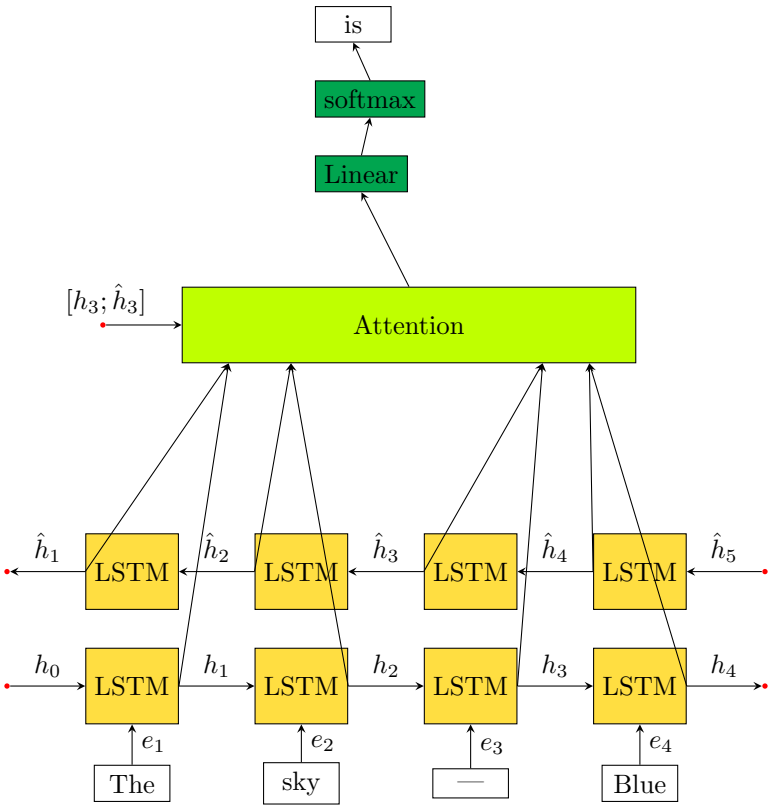


Figure 4: Bidirectional Masked Language Model with Attention

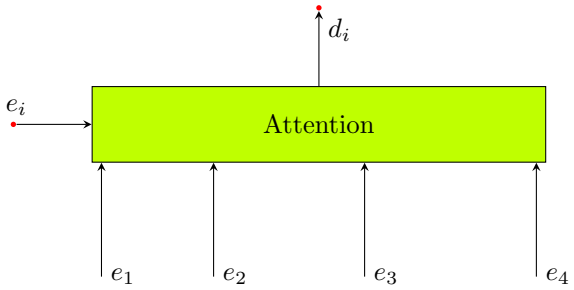


Figure 5: Selft Attention block diagram, i is i^{th} word, e_i is the embedding of $word_i$

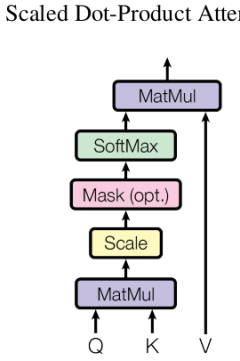


Figure 6: Attention Block of Transformer.
$Q = \begin{bmatrix} e_1 W_q & e_2 W_q & \cdots e_i W_q \cdots \end{bmatrix}^T$, $K = \begin{bmatrix} e_1 W_k & e_2 W_k & \cdots e_i W_k \cdots \end{bmatrix}^T$, $V = \begin{bmatrix} e_1 W_v & e_2 W_v & \cdots e_i W_v \cdots \end{bmatrix}^T$, e_i is the i^{th} embedding of the $token_i$, W_q, W_k, and W_v are projection matrices to equate dimentions of the dot product QK^T, This form is similar to the above equations 1.1

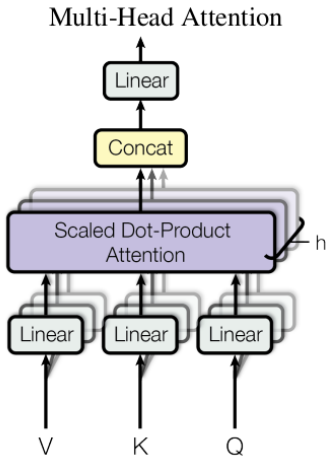


Figure 7: Multi Head self Attention.

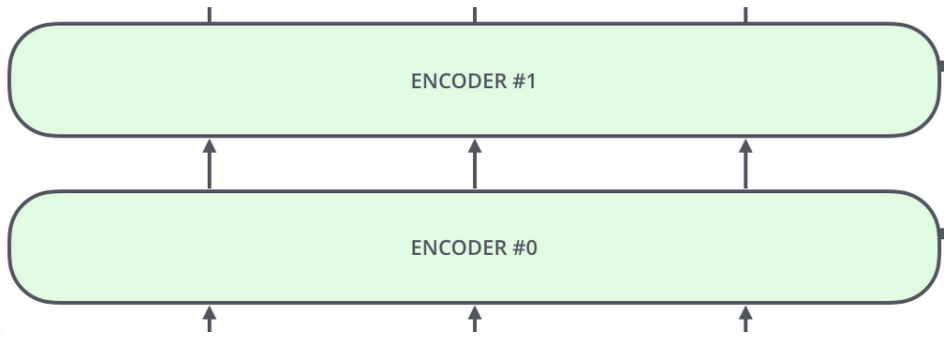


Figure 8: The full Encoder of the Transformer.
Every $Encoder_i$ has (A) Heads (i.e: "Self Attention" blocks) as in Figure 7

- Next Sentence Prediction (NSP): Given two sentences BERT is trained to answer this simple question: "Is the second sentence next the first sentence?"

Our Contribution involves:

- Representing True bidirectional Language Modeling, not previous models that contaminates Left to Right (LTR), and Right To Left (RTL) Language Models.
- our model by fine-tuning does achieves state of the art results, so our model reduces the need for task-specific architecture.

3 Related Work

The Litratue is Long we will focus on the preivous State of the Art. We have 3 categories: finetuing, feature bases, transer learning.

3.1 Unsupervised Feature-based Approach

The previous state of the art was ElMo [SWWP+21], and their predecessors uses two separate language models (i.e: LSTMs) of for the LFT, and the other for RTL, then concatinate both. That is not full bidirectional in nature but achibed state of the art in QA, Sentiment Classification, and Name Entity Recognition.

3.2 Unsupervised Fine-tuning Approach

The previous state of the art was Open AI GPT [RNSS18] that uses uinidirectionl RTL Encoder of the transformer, and achieved the state of the art on stanfod SQUAD data set. The issue with this is that it limits the LM ability to learn full context of the language. Also it added sparse in training the model.

3.3 Transfer Learning from Supervised data

There were no NLP in the litrature of the paper that uses transfer Learning.

4 BERT

Here will talk about the model architecture at both: pre-training ,and fine-tuning.

4.1 Model Architecture

The model is the Encoder block of the Transformer [VSP+17]. as shown in Figure 9 every output token T_i has attention on all the input tokens E_i.

We donate (L) as the number of encoders, (A) as the number of heads, (H) the hidden vector size. We have two models:

- $BERT_{base}$: with(L=12, H=768, A=12, with total parameters=110M).
- $BERT_{large}$: with (L=24, H=1024, A=16, with total parameters=340M).

We introduced $BERT_{base}$ to be comparable with Open AI GPT as they share nearly the same architecture.

4.2 Input/Output Representation

4.2.1 BERT Embedding

The input input embedding to BERT is well explained at Figure 10

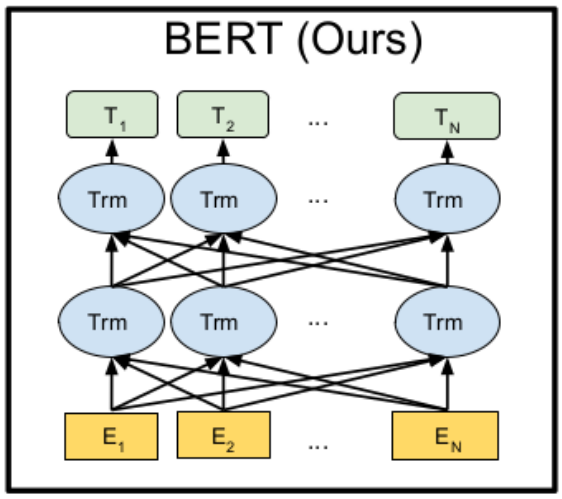


Figure 9: General Overview of the BERT (i.e: Encoder of the Transformer).

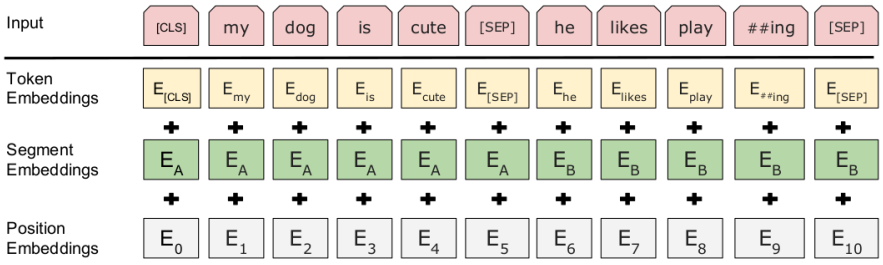


Figure 10: The Emending of BERT.
The first row is input embedding form WordPiece which contains 300,000 words with three special tokens optimized throw the training: `[cls]` (class) token which donates start of the sequence, `[sep]` (separator) which placed at the end of every sentence, `[mask]` donates the masked word. **The second row** has two embeddings: E_A, E_B they discriminates sentenceA form sentenceB. **The last row** embedding is the positional embedding form the transformers to decode the positron for every word. All the embedding are added together to represent the input to our BERT model.

4.2.2 Some BERT Definitions

- **Sentence**: not the actual linguistic sentence, it is defined here as a contiguous number of words.
- **Sequence**: the input to the BERT model that contain sentence or more.
- `[cls]` is the (class) token that represnets the beginig of the sequecne. The token is optimized in the training the output corresponding to it is $C \in \mathbb{R}^H$ as in Figure 11.
- `[sep]` is a (separator) token which indicates end of sentence.
- `[mask]` is the mask to hide a word in order to make the model predict it.

4.3 Pre-traing BERT

4.4 Masked Language Model

As we said we used Masked LM to training our model we added `[mask]` token to represents the hidden word that we hope the model will predict it. But we have a problem the `[mask]` token exists only at

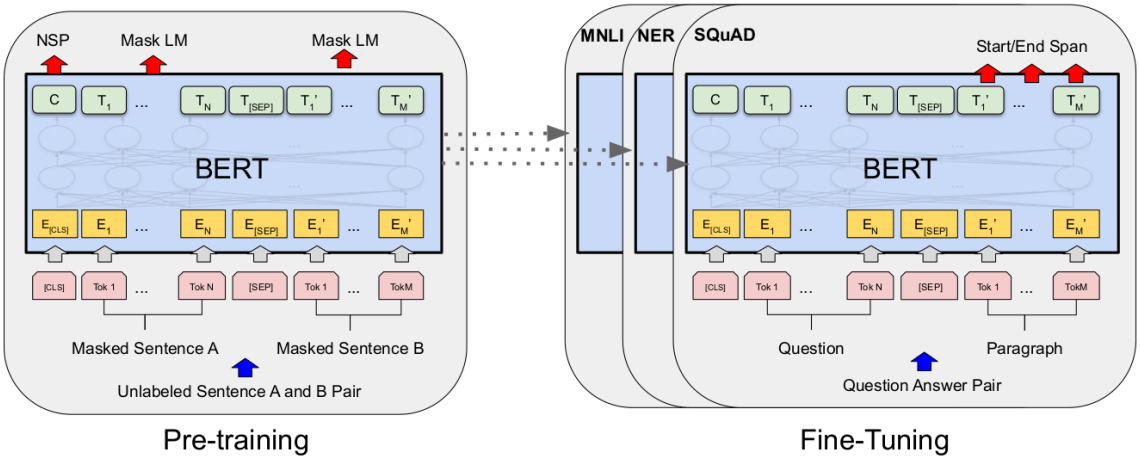


Figure 11: (at the left) The BERT Model for pre-training, (at the right) BERT fine-tuned tasks.

traing not at fine-tuning what shall we do?
We will mask 15% of the input sequence the masked words will be replaced by:

- the `[mask]` token 80% of the time
- a random word (i.e: token) 10% of the time.
- the word itself 10% of the time.

For example from Appendix A1 from the paper:
Assuming the unlabeled sentence is `my dog ishairy` , and during the random masking procedure we chose the 4^{th} token (which corresponding to `hairy`), our masking procedure can be further illustrated by:

- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy → my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy → my dog is apple`.
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy → my dog is hairy` . The purpose of this is to bias the representation towards the actual observed word.

4.4.1 Next Sentence Prediction

In many application our input has more than one sentence like in Question Answering (QA), or in Natural Language Infrenceing (NLI). To solve this we train a barbarized next sentence predictor (NSP) that predicts whether the SentenceB follows SentenceA or not. At training we put:

- 50% of the time SentenceB is actually following sentenceA with labeled as *IsNext*.
- 50% of the time SentenceB is **NOT** following sentenceA with labeled as *NotNext*.

This binary classification is done throw the output token (C) the corresponds to input token `[cls]` as in Figure 11.

4.5 Fine-Tuning

Fine-tuing is simple we first replace the BERT input/output by our downstream task, then we train the whole model as End to End. This takes small time (i.e: 1 hour on single google TPU, or few hours using GPUs). **While manipulating input**: we have two sentences: `sentenceA`, `sentenceB` to the BERT input. The `sentenceA`, `sentenceB` are:

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Figure 12: The Results of Glue Benchmark for every dataset, and the average for all datasets.

- Sentencing paris in praphrasing task.
- hypothesis, and premises (e.g: Mohamed goes to school by bike, Mohamed can ride bike) in instalment task.
- question, and sentence that have the answer in Question Answer Task.
- input sentence, and degenerate $text\phi$ in text classification tasks.

While manipulating output: we have two categories of downstream tasks.

- Binary Classification: (like Sentiment classification, or entailment classification) we use the output (C) that corresponds to `[cls]` token as in Figure 11.
- Downstream Tasks that involves manipulating more than one output token (T_i) like; Question Answer in Figure 11.

4.6 Pre-training Data

"For the pre-training corpus we use the BooksCorpus [ZKZ+15] (800M words) and English Wikipedia (2,500M words). For Wikipedia we extract only the text passages and ignore lists, tables, and headers. It is critical to use a document-level corpus rather than a shuffled sentence-level corpus such as the Billion Word Benchmark in order to extract long contiguous sequences."

5 Experiments

Here we show BERT fine-tuing on 11 NLP tasks.

5.1 GLUE

The General Understanding Evaluation Benchmark (GLUE) a set of several data sets of different classification tasks. We used $C \in \mathbb{R}^H$ that corresponding the aggregate represntation for the input sequecne, and a classficatin layer $W \in \mathbb{R}^{KxH}$, where K is the number of classes. We used standard classification loss i.e: $log(softmax(CW^T))$.

5.1.1 GLUE Hyper parameters

- epoches: 3
- batch size: 32
- learning rate: among($5e-5$, $4e-5$,$3e-5$,and $2e-5$)

Because BERT in general converges slower, so we need more training steps. On small datasets traaining was unstable with $BERT_{large}$, so we made more than one training with random restarts using same checkpoint (i.e: continue form last random restat) with different shuffling, and different classifier layer initialization. The result are in Figure 12.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
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 13: The Results of SQuAd v1.1.
EM stands for Exact Match is an evaluation standard the dataset delivers which means "Measures the percentage of prediction that matches the ground truth label"

5.2 SQuAD v1.1

The Stanford Question Answering Dataset (SQuAD v1.1) is a collection of 100k crowd-sourced question/answer pairs. In QA we have two sentnces:

- The question Sentence, that will be `sentenceA` in BERT.
- The Answer Sentence that have the answer, that will be `sentenceB` in BERT.

Our goal is to find the start and the end of the answer in the answer sentence.

5.2.1 Fin-tuing Procedure

As we goal is to find the start of answer token $T_i \in \mathbb{R}^H$, and end of answer token $T_i \in \mathbb{R}^H$. We introduce start vector ($S \in \mathbb{R}^H$, then we do product S with every output token of `sentenceB` T_i, then we take softmax among words in `sentenceB` as in equation 5.2.1

$$P_i^{start} = \frac{e^{S.T_i}}{\sum_j e^{S.T_j}}$$

Similar to start we introduce end vector ($E \in \mathbb{R}^H$, the end of answer is calculated as in Equation 5.2.1

$$S.T_i + E.T_j, \quad wherej > i$$

where i is the start token, j is the end. The loss is the log-likelihood of correct start and end positrons

5.2.2 SQuAD v1.1 Hyper parameters

- epoches: 3
- batch size: 32
- learning rate: $5e-5$

Results are shown in Figure 13. Note that SQuAD allows for models to use more than training data, so we fine-tun first on TriviaQA, then of SQuAD.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

Figure 14: The Results of SQuAd v2.0.
EM stands for Exact Match is an evaluation standard the dataset delivers which means "Measures the percentage of prediction that matches the ground truth label"

5.3 SQuAD v2.0

The data set is the same as SQuAD v1.1 but with a no answer. We solve the no answer as the the start and the end of the anseer at `[cls]` token. For Prediction we compare $s_{null} = S.C = E.C$ (null naswer span) with $\hat{s} = max_{j>i}(S.T_i + E.T_j)$ (answer span). We predict a none null answer if $\hat{s} > s_{null} + \tau$.

5.3.1 SQuAD v2.0 Hyper parameters

- epoches: 2
- batch size: 48
- learning rate: $5e-5$

Results are shown in Figure 14. Note we did not here more than data-set we only used SQuAD v2.0.

5.4 SWAG

The Situations With Adversarial Generations (SWAG) dataset contains 113k sentence-pair completion examples that evaluate grounded common-sense inference. Given an input sentence the task is to find the most suitable answer among 4 choices. We fin-tune by: construct four ouput vector each one has input of:

- `sentenceA` is the input sentence
- `sentecneB` is one of the four choices

The ouput of every choice is (C) (i.e: output of `[cls]`). The only task specific parameter we introduce is a vector ($V \in \mathbb{R}^H$). We dot product V with C for every choice, Then we take softmax with the four choices.

5.4.1 SWAG Hyper parameters

- epoches: 3
- batch size: 16
- learning rate: $2e-5$

Results are shown in Figure 15.

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Figure 15: The Results of SWAG
Human means here the performance of human no models, jut normal humans.

	Dev Set				
Tasks	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

Figure 16: Trying different version of $BERT_{base}$ with different variation
NSP donates "Next Sentence Prediction", LTR is a standard "Left to Right language model", BILSTM is a "bidirectional LSTM" which concatenates Left to Right LSTM with Right to left LSTM.

6 Ablation Studies

We further look at some details at this section to better understand BERT

6.1 Effect of Pre-training Tasks

Here We examin $BERT_{base}$ with removing our contribution: Next Snetence Prediction (NSP), Bidirectional Maked Language Model (MLM) then we evaluate on some of GLUE benckmark tasks, and SQuAD v1.1.

No NSP: We our bidirectional MLM without NSP.

LTR & NSP here we use the standrd Left to Right (LTR) Language Model (LM) rather than MLM (`i.e: the sky is --`). This directly same as Open AI GPT, but trained on more data and different fin-tuning. This implementation has many downsides: "Creating pre-train/fine-tune mismatch that degrades performance" `-- not understood`.

The results are in Figure 16, ws see that using LTR LM degrades performance significantly. we can see that BiLSTM is much better than LTR approach to solve QA problem in SQuAD. But BiLSTM like ELMo [SWWP+21] has many downsides:

- twice expensive as a single bidirectional model.
- that is less powerful than our deep bidirectional model becasue it uses the whole context in all layers.

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 17: Training different model sizes
L is the number of encoders, H is the lenth of the hidden vector, A is the number of heads for the Transformer. ppl is the perplexity (e^{loss})

6.2 Effect of Model Size

In this section we analyse the effect of model size by these assumptions:

- We used the same hyper parameters across all tests.
- we report the 5 random restart of fine-tuning.

We use to know that using bigger model on small dateset will under-fit that model, so performance will degrade. But here we show that using larger model improves the accuracy evenv with small dataset like MPRC with 3,600 samples why ? this due to the pre-training process we used, we pre-trained our model with 3,300M wrod (our model has learned before this is only fine-tuning). Results are reported in Figure 17. If increasing model size improves accuracy why not use larger model ? after that that does not improve accuracy that much.

6.3 Feature-based approach with BERT

Here we take ouput tokens as the input as a feature (input of other model) that have two advantages:

- Improves computation as we only focus on our downstream task architecture not train
- not all tasks can be applied using transforms encoder using feature-based approach gives us freedom to chose the model.

In this section, we compare the two approaches by applying BERT to the CoNLL-2003 Named Entity Recognition (NER) task. In the input to BERT, we use a case-preserving WordPiece model, and we include the maximal document context provided by the data. Following standard practice, we formulate this as a tagging task but do not use a CRF layer in the output. We use the representation of the first sub-token as the input to the token-level classifier over the NER label set we used biLSTM with hiden size of=768. Results are in Figure 18.

7 Conclusion

We introduced a generalized model that can be used in many NLP tasks which proves the benefit of using transfer learning in language models, and bidirectional context representation. Even if we fine-tune our model on small datasets we can get high performance thank to pre-training.

We organize the appendix into three sections:

- Additional implementation details for BERT are presented in Appendix A;
- Additional experiment at Appendix B;
- Additional ablation studies in Appendix C;

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Figure 18: CoNLL-2003 Named Entity Recognition results
Hyper-parameters were selected using the Devset. The reported Dev and Test scores are averaged over 5 random restarts using those hyper-parameters.

A Additional Details of BERT

A.1 Masked Language model Procedure

explained in Masked Language Model Section 4.4.

A.2 Pre-training Procedure [copied]

To generate each training input sequence, we sample two spans of text from the corpus, which we refer to as "sentences" even though they are typically much longer than single sentences (but can be shorter also). The first sentence receives the `A` embedding and the second receives the `B` embedding. 50% of the time `B` is the actual next sentence that follows A and 50% of the time it is a random sentence, which is done for the "next sentence prediction" task. They are sampled such that the combined length is $\leq$ 512 tokens. The LM masking is applied after WordPiece tokenization with a uni- form masking rate of 15%, and no special consideration given to partial word pieces. We train with batch size of 256 sequences (256 sequences * 512 tokens = 128,000 tokens/batch) for 1,000,000 steps(number of step weighs will be updated), which is approximately 40 epochs over the 3.3 billion word corpus. We use Adam with learning rate of $1e-4$, $\beta_1 = 0.9, \beta_2 = 0.999$, L2 weight decay of 0.01, learning rate warmup over the first 10,000 steps, and linear decay of the learning rate. We use a dropout prob- ability of 0.1 on all layers. We use a `gelu` activation [HG16] rather than the standard `relu`, following OpenAI GPT. The training loss is the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood. Training of $BERT_{BASE}$ was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total). 13 Training of $BERT_{LARGE}$ was performed on 16 Cloud TPUs (64 TPU chips total). Each pre-training took 4 days to complete. Longer sequences are disproportionately expensive because attention is quadratic to the sequence length. To speed up pretraing in our experiments, we pre-train the model with sequence length of 128 for 90% of the steps. Then, we train the rest 10% of the steps of sequence of 512 to learn the positional embeddings.

A.3 Fine-tuning Procedure

Hyper-parameters:

- Dropout probabilty: 0.1
- Batch Size: 16, 32
- Learning Rate (Adam): $5e-5$, $3e-5$, $2e-5$
- Number of Epochs: 2, 3, 4

BERT	Open AI GPT	ELMo
bidirectional	LTR	LTR concatenated with RTL
based on Transformer Encoder	based on Transformer Encoder	RTL LSTM with RTL LSTM
trained on book corpus 800M and wikipedia 2,500M	trained on book corpus 800M only	
`[cls]`, and `[sep]` are trained and used in fine-tuning	`[cls]`, and `[sep]` are introduced in fine-training only not at training	
trained with 1M steps with batch size of 128,000 words	trained with 1M steps with batch size of 32,000 words	

Table 1: BERT vs Open AI GPT vs ELMo.

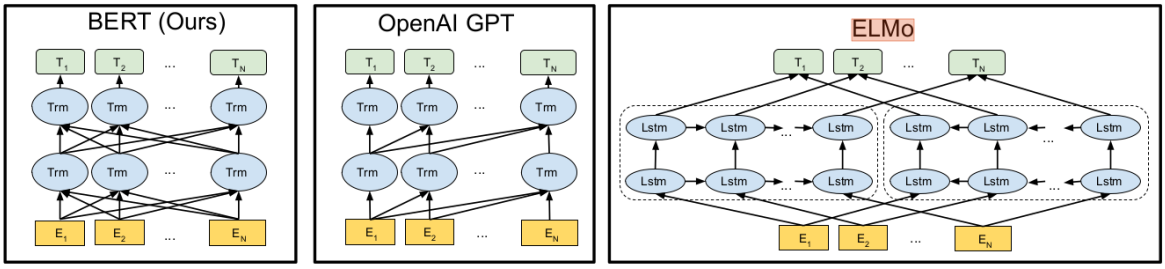


Figure 19: BERT vs Open AI GPT vs ELMo architectures

A.4 Comparison BERT, ELMo, and Open AI GPT

This Table 1 and Figure 19 illustrate the difference of Architecture between the three models.

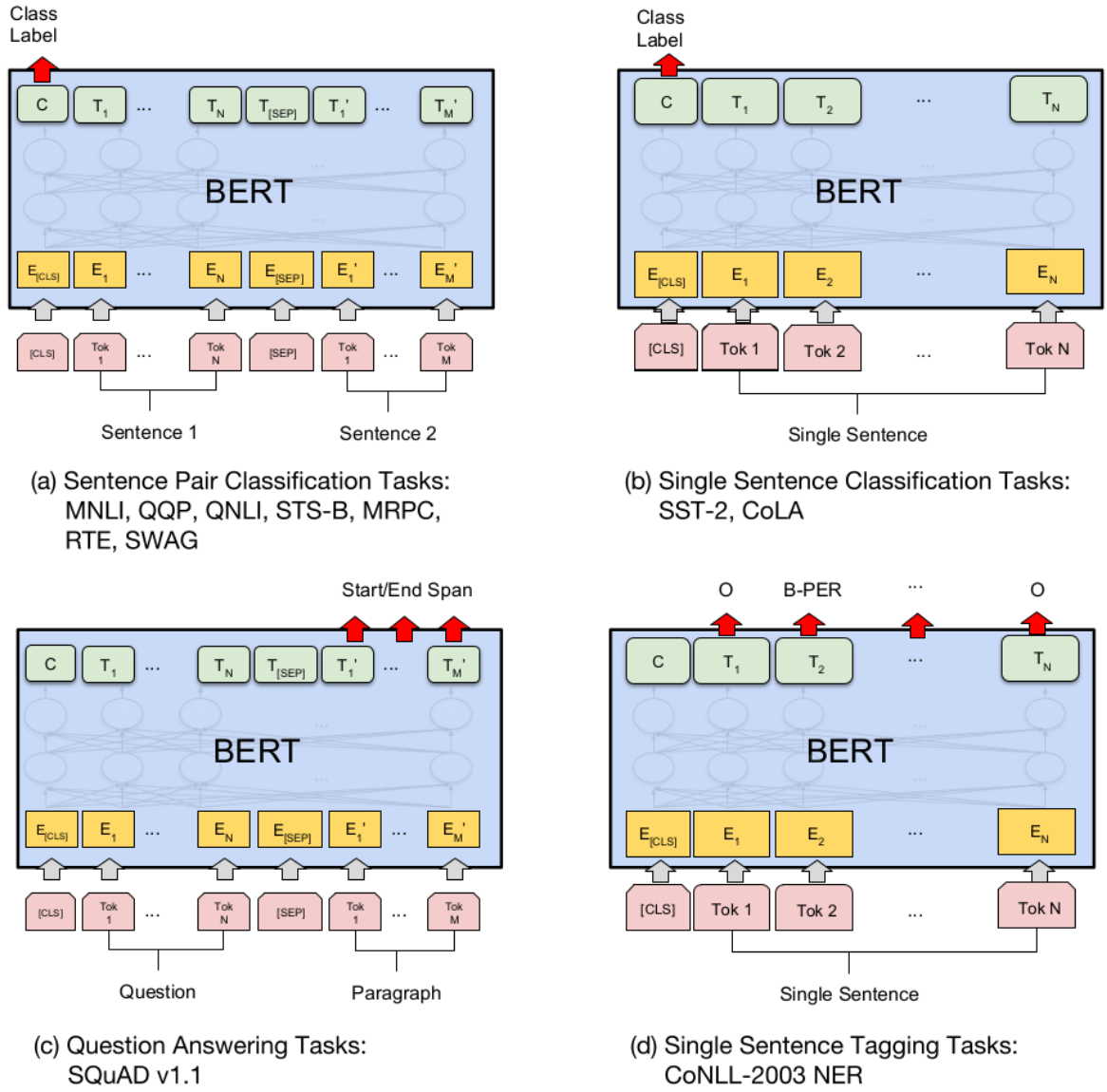
A.5 Illustrations of Fine-tuning on Different Tasks

Figure 20 shows using BERT for different downstream tasks.

B Detailed Experiment Step

B.1 GLUE Benchmark [copied]

Just a description of GLUE benchmark the datasets to not big deal to illustrate.
MNLI Multi-Genre Natural Language Inference is a large-scale, crowdsourced entailment classification task. Given a pair of sentences, the goal is to predict whether the sec- ond sentence is an entailment, contradiction, or neutral with respect to the first one.
QQP Quora Question Pairs is a binary classification task where the goal is to determine if two questions asked on Quora are semantically equivalent .
QNLI Question Natural Language Inference is a version of the Stanford Question Answering Dataset which has been converted to a binary classification task. The positive examples are (question, sentence) pairs which do contain the correct answer, and the negative examples are (question, sentence) from the same paragraph which do not contain the answer.
SST-2 The Stanford Sentiment Treebank is a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment.
CoLA The Corpus of Linguistic Acceptability is a binary single-sentence classification task, where the goal is to predict whether an English sentence is linguistically “acceptable” or not.



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

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

(c) Question Answering Tasks: SQuAD v1.1

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

Figure 20: BERT fine-tunded with different tasks

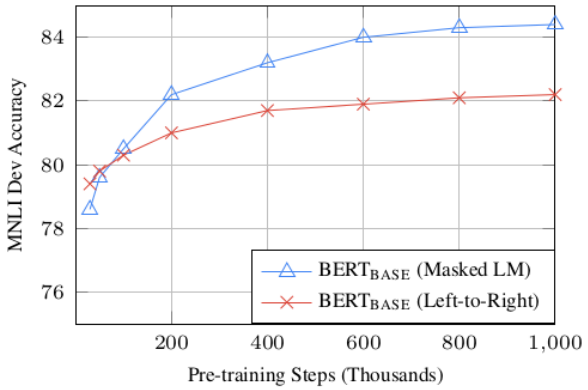


Figure 21: LTR BERT vs MLM BERT at the begining of training

STS-B The Semantic Textual Similarity Benchmark is a collection of sentence pairs drawn from news headlines and other sources. They were annotated with a score from 1 to 5 denoting how similar the two sentences are in terms of semantic meaning.
MRPC Microsoft Research Paraphrase Corpus consists of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent
RTE Recognizing Textual Entailment is a binary entailment task similar to MNLI, but with much less training data.
WNLI Winograd NLI is a small natural language inference dataset. The GLUE webpage notes that there are issues with the construction of this dataset, and every trained system that's been submitted to GLUE has performed worse than the 65.1 baseline accuracy of predicting the majority class. We therefore exclude this set to be fair to OpenAI GPT. For our GLUE submission, we always predicted the majority class.

C Additional Ablation Studies

C.1 Effect of Number of Training Steps [copied]

Figure 21 presents MNLI Dev accuracy after fine-tuning from a checkpoint that has been pre-trained for k steps. This allows us to answer the following questions:

1. Question: Does BERT really need such a large amount of pre-training (128,000 words/batch * 1,000,000 steps) to achieve high fine-tuning accuracy?
   Answer: Yes, BERT BASE achieves almost 1.0% additional accuracy on MNLI when trained on 1M steps compared to 500k steps.

2. Question: Does MLM pre-training converge slower than LTR pre-training, since only 15% of words are predicted in each batch rather than every word?
   Answer: The MLM model does converge slightly slower than the LTR model. However, in terms of absolute accuracy the MLM model begins to outperform the LTR model almost immediately.

C.2 Ablation Different Masking Procedures

Here at Figure 22. we found that feature based approach is robust for masking procedures.

References

[HG16] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. 2016.

Masking Rates			Dev Set Results		
MASK	SAME	RND	MNLI	NER	
			Fine-tune	Fine-tune	Feature-based
80%	10%	10%	84.2	95.4	94.9
100%	0%	0%	84.3	94.9	94.0
80%	0%	20%	84.1	95.2	94.6
80%	20%	0%	84.4	95.2	94.7
0%	20%	80%	83.7	94.8	94.6
0%	0%	100%	83.6	94.9	94.6

Figure 22: Diffident masking Procedures
Same means: puting the masked word the same word, Mask means: replacing the masked word with token `[mask]`, RND means: replacing the masked word with a random word form the corpus.MNLI stands for Masked Natural Language Inferencing, NER: Named Entity Recognition.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.

[SWWP+21] Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135, 2021.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[ZKZ+15] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.