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 (Peters et al., 2018; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT representations 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 benchmark to **80.4%** (**7.6%** absolute improvement), MultiNLI accuracy to **86.6%** (**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**.

1 Introduction

Language model pre-training has shown to be effective for improving many natural language processing tasks (Dai and Le, 2015; Peters et al., 2017, 2018; Radford et al., 2018; Howard and Ruder, 2018). These tasks include sentence-level tasks such as natural language inference (Bowman et al., 2015; Williams et al., 2018) and paraphrasing (Dolan and Brockett, 2005), which aim to predict the relationships between sentences by analyzing them holistically, as well as token-level tasks such as named entity recognition (Sang and Meulder, 2003) and SQuAD question answering (Rajpurkar et al., 2016), where models are required to produce fine-grained output at the token-level.

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018), uses tasks-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning the pre-trained parameters. In previous work, both approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

We argue that current techniques severely restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attended to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restric-

tions are sub-optimal for sentence-level tasks, and could be devastating when applying fine-tuning based approaches to token-level tasks such as SQuAD question answering (Rajpurkar et al., 2016), where it is crucial to incorporate context from both directions.

In this paper, we improve the fine-tuning based approaches by proposing BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. BERT addresses the previously mentioned unidirectional constraints by proposing a new pre-training objective: the "masked language model" (MLM), inspired by the Cloze task (Taylor, 1953). The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre-training, the MLM objective allows the representation to fuse the left and the right context, which allows us to pre-train a deep bidirectional Transformer. In addition to the masked language model, we also introduce a "next sentence prediction" task that jointly pre-trains text-pair representations.

The Contributions of our paper are as follows:

- We demonstrate the importance of bidirectional pre-training for language representations. Unlike Radford et al. (2018), which uses unidirectional language models for pre-training, BERT uses masked language models to enable pre-training deep bidirectional representations. This is also in contrast to Peters et al. (2018), which uses a shallow concatenation of independently trained left-to-right and right-to-left LMs.
- We show that pre-trained representations eliminate the needs of many heavily-engineered task-specific architectures. BERT is the first fine-tuning based representation model that achieves state-of-the-art performance on a large suite of sentence-level *and* token-level tasks, outperforming many systems with task-specific architectures.
- BERT advances the state-of-the-art for eleven NLP tasks. We also report extensive ablations of BERT, demonstrating that the bidirectional nature of our model is the single most important new contribution. The code and pre-trained model will be available at `goo.gl/language/bert`.

2 Related Work

There is a long history of pre-training general language representations, and we briefly review the most popular approaches in this section.

2.1 Feature-based Approaches

Learning widely applicable representations of words has been an active area of research for decades, including non-neural (Brown et al., 1992; Ando and Zhang, 2005; Blitzer et al., 2006) and neural (Collobert and Weston, 2008; Mikolov et al., 2013; Pennington et al., 2014) methods. Pre-trained word embeddings are considered to be an integral part of modern NLP systems, offering significant improvements over embeddings learned from scratch (Turian et al., 2010).

These approaches have been generalized to coarser granularities, such as sentence embeddings (Kiros et al., 2015; Logeswaran and Lee, 2018) or paragraph embeddings (Le and Mikolov, 2014). As with traditional word embeddings, these learned representations are also typically used as features in a downstream model.

ELMo (Peters et al., 2017) generalized traditional word embedding research along a different dimension. They propose to extract *context-sensitive* features from a language model. When integrating contextual word embeddings with existing task-specific architectures, ELMo advances the state-of-the-art for several major NLP benchmarks (Peters et al., 2018) including question answering (Rajpurkar et al., 2016) on SQuAD, sentiment analysis (Socher et al., 2013), and named entity recognition (Sang and Meulder, 2003).

2.2 Fine-tuning Approaches

A recent trend in transfer learning from language models (LM) is to pre-train some model architecture on a LM objective before fine-tuning that same model for a supervised downstream task (Dai and Le, 2015; Howard and Ruder, 2018; Radford et al., 2018). The advantage of these approaches is that few parameters need to

be learned from scratch. At least partly due this advantage, OpenAI GPT (Radford et al., 2018) achieved previously state-of-the-art results on many sentence-level tasks from the GLUE benchmark (Wang et al., 2018).

2.3 Transfer Learning from Supervised Data

While the advantage of unsupervised pre-training is that there is a nearly unlimited amount of data available, there has also been work showing effective transfer from supervised tasks with large datasets, such as natural language inference (Conneau et al., 2017) and machine translation (McCann et al., 2017). Outside of NLP, computer vision research has also demonstrated the importance of transfer learning from large pre-trained models, where an effective recipe is to fine-tune models pre-trained on ImageNet (Deng et al., 2009; Yosinski et al., 2014).

3 BERT

We introduce BERT and its detailed implementation in this section. We first cover the model architecture and the input representation for BERT. We then introduce the pre-training tasks, the core innovation in this paper, in Section 3.3. The pre-training procedures, and fine-tuning procedures are detailed in Section 3.4 and 3.5, respectively. Finally, the differences between BERT and OpenAI GPT are discussed in Section 3.6.

3.1 Model Architecture

BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in (Vaswani et al., 2017) and released in the `tensor2tensor` library [1]. Because the use of Transformers has become ubiquitous recently and our implementation is effectively identical to the original, we will omit an exhaustive background descriptioin of the model architecture and refer readers to Vaswani et al. (2017) as well as excellent guides such as "The Annotated Transformer."[2] In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A. In all cases we set the feed-forward/filter size to be $4H$, i.e., 3072 for the $H = 768$ and 4096 for the $H = 1024$. We primarily report results on two model sizes:

- **BERT$_{\text{BASE}}$** L=12, H=768, A=12, Total Parameters=110M
- **BERT$_{\text{LARGE}}$**: L=24, H=1024, A=16, Total Parameters=340M

BERT$_{\text{BASE}}$ was chosen to have an identical model size as OpenAI GPT for comparison purposes. Critically, however, the BERT Transformer uses bidirectional self-attention, while the GPT Transformer uses constrained self-attention where every token can only attend to context to its left. We not that in the literature the bidirectional Transformer is often referred to as a "Transformer encoder" while the left-context-only version is referred to as a "Transformer decoder" since it can be used for text generation. The comparisons between BERT, OpenAI GPT and ELMo are shown visually in Figure 1.

3.2 Input Representation

Our input representation is able to unambiguously represent both a single text sentence or a pair of text sentences (e.g., [Question, Answer]) in one token sequence [3]. For a give token, its input representation is constructed by summing the corresponding token, segment and position embeddings. A visual representation of our input representation is give in Figure 2.

The specifics are:

- We use WordPiece embeddings (Wu et al., 2016) with a 30000 token vocabulary. We denote split word pieces with ##.
- We use learned positional embeddings with supported sequence lengths up to 512 tokens.
- The first token of every sequence is always the special classification embedding

[1] https://github.com/tensorflow/tensor2tensor

[2] http://nlp.seas.harvard.edu/2018/04/03/attention.html

[3] Throughout this work, a "sentence" can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. A "sequence" refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together.

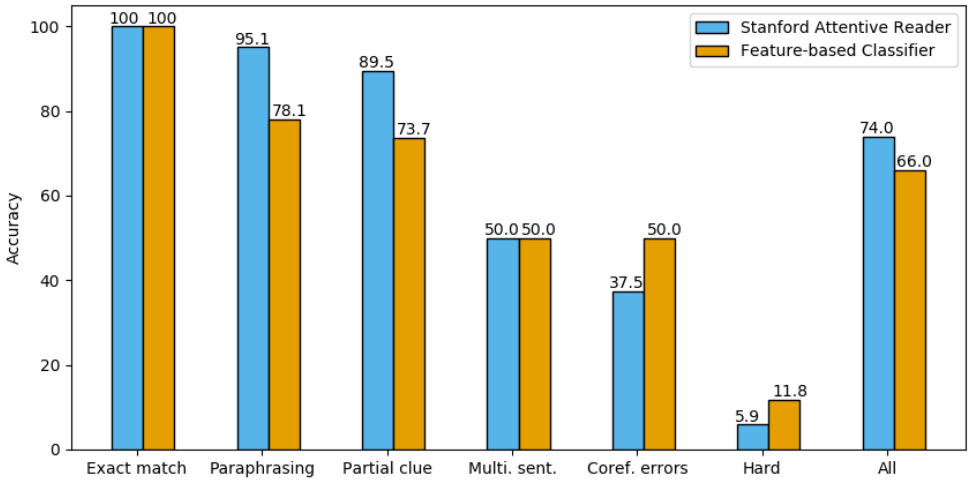

Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right- to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

([CLS). The final hidden state (i.e., output of Transformer) corresponding to this token is used as the aggregate sequence representation for classification tasks. For non-classification tasks, this vector is ignored.

- Sentence pairs are packed together into a single sequence. We differentiate the sentences in two ways. First, we separate them with a special token ([SEP]). Second, we add a learned sentence A embedding to every token of the first sentence and a sentence B embedding to every token of the second sentence.
- For single-sentence inputs we only use the sentence A embeddings.

3.3 Pre-training Tasks

Unlike Peters et al. (2018) and Radford et al. (2018), we do not use traditional left-to-right or right-to-left language models to pre-train BERT. Instead, we pre-train BERT using two novel unsupervised prediction tasks, described in this section.

3.3.1 Task #1: Masked LM

Intuitively, it is reasonable to believe that a deep bidirectional model is strictly more powerful than either a left-to-right model or the shallow concatenation of a left-to-right and right-to-left model. Unfortunately, standard conditional language models can only be trained left-to-right *or* right-to-left, since bidirectional conditioning would allow each word to indirectly "see itself" in a multi-layered context.

In order to train a deep bidirectional representation, we take a straightforward approach of masking some percentage of the input tokens at random, and then predicting only those masked tokens. We refer to this procedure as a "masked LM"(MLM), although it is often referred to as a *Cloze* task in the literature (Taylor, 1953). In this case, the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM. In all of our experiments, we mask 15% of all WordPiece tokens in each sequence at random. In contrast to denoising auto-encoders (Vincent et al., 2008), we only predict the masked words rather than reconstructing the entire input.

Although this does allow us to obtain a bidirectional pre-trained model, there are two downsides to such an approach. The first is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning. To mitigate this, we do not always replace "masked" words with the actual

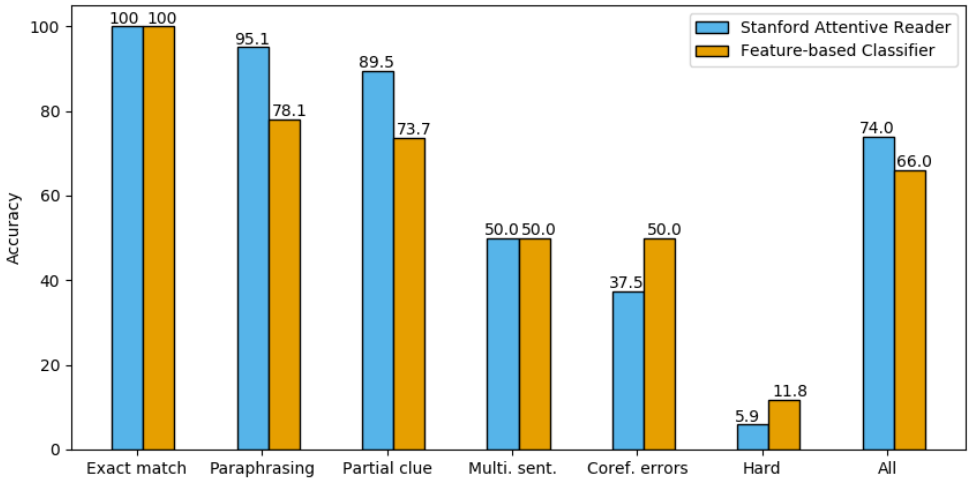

Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

[MASK] token. Instead, the training data generator chooses 15% of tokens at random, e.g., in the sentence my dog is hairy it chooses hairy. It then performs the following procedure:

- Rather than *always* replacing the chosen words with [MASK], the data generator will do the following:
- 80% of the time: Replace the word with [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.

The Transformer encoder does not know which words it will be asked to predict or which have been replaced by random words, so it is forced to keep a distributional contextual representation of *every* input token. Additionally, because random replacement only occurs for 1.5% of all tokens (i.e., 10% of 15%), this does not seem to harm the model's language understanding capability.

The second downside of using an MLM is that only 15% of tokens are predicted in each batch, which suggests that more pre-training steps may be required for the model to converage. In Section 5.3 we demonstrate that MLM does converge marginally slower than a left-to-right model (which predicts every token), but the empirical improvements of the MLM model far outweigh the increased training cost.

3.3.2 Task #2: Next Sentence Prediction

Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the *relationship* between two text sentences, which is not directly captured by language modeling. In order to train a model that understands sentence relationship, we pre-train a binarized *next sentence prediction* task that can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences A and B for each pre-training example, 50% of the time B is the actual next sentence that follows A, and 50% of the time it is a random sentence from the corpus. For example:

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

Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

We choose the NotNext sentences completely at random, and the final pre-trained model

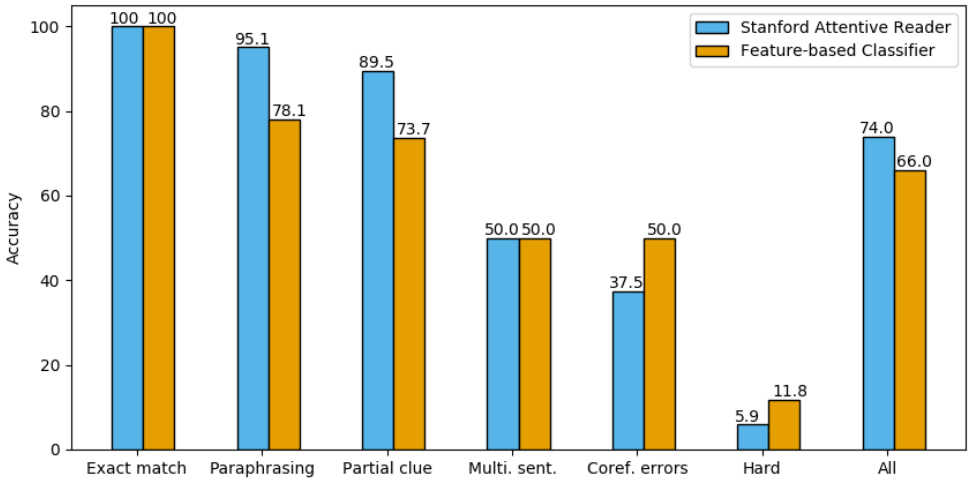

Figure 3: Our task specific models are formed by incorporating BERT with one additional output layer, so a minimal number of parameters need to be learned from scratch. Among the tasks, (a) and (b) are sequence-level tasks while (c) and (d) are token-level tasks. In the figure, E represents the input embedding, Ti represents the contextual representation of token i, [CLS] is the special symbol for classification output, and [SEP] is the special symbol to separate non-consecutive token sequences.

achieves 97%-98% accuracy at this task. Despite its simplicity, we demonstrate in Section 5.1 that pre-training towards this task is very beneficial to both QA and NLI.

3.4 Pre-training Procedure

The pre-training procedure largely follows the existing literature on language model pre-training.

For the pre-training corpus we use the concatenation of BooksCorpus (800M words) (Zhu et al., 2015) and English Wikipedia (2500M 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 (Chelba et al., 2013) in order to extract long contiguous sequences.

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 uniform 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 = 128000 tokens/batch) for 1000000 steps, 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 10000 steps, and linear decay of the learning rate. We use a dropout probability of 0.1 on all layers. We use a gelu activation (Hendrycks and Gimpel, 2016) rather than the standard relue, following OpenAI GPT. The training loss is the sum of the mean masked LM likelihood and mean next sentence prediction likelihood.

Training of $\text{BERT}_{\text{BASE}}$ was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total)[4]. Training of $\text{BERT}_{\text{LARGE}}$ was performed on 16 Cloud TPUs (64 TPU chips total). Each pre-training took 4 days to complete.

[4]https://cloudplatform.google-blog.com/2018/06/Cloud-TPU-now-offers-preemptible-pricing-and-global- availability.html

3.5 Fine-tuning Procedure

For sequence-level classification tasks, BERT fine-tuning is straightforward. In order to obtain a fixed-dimensional pooled representation of the input sequence, we take the final hidden state (i.e., the output of the Transformer) fo the first token in the input, which by construction corresponds to the special [CLS] word embedding. We denote this vector as $C \in \mathbb{R}^H$. The only new parameters added during fine-tuning are for a classification layer $W \in \mathbb{R}^{K*H}$, where K is the number of classifier labels. The label probabilities $P \in \mathbb{R}^K$ are computed with a standard softmax, $P = softmax(CW^T)$. All of the parameters of BERT and W are fine-tuned jointly to maximize the log-probability of the correct label. For span-level and token-level prediction tasks, the above procedure must be modified slightly in a task-specific manner. Details are given in the corresponding subsection of Section 4.

For fine-tuning, most model hyperparameters are the same as in pre-training, with the exception of the batch size, learning rate, and number of training epochs. The dropout probability was always kept at 0.1. The optimal hyperparameter values are task-specific, but we found the following range of possible values to work well across all tasks:

- **Batch size**: 16, 32
- **Learning rate (Adam)**: 53-5, 3e-5, 2e-5
- **Number of epochs**: 3, 4

We also observed that large data sets (e.g., 100k+ labeled training examples) were far less sensitive to hyperparameter choice than small data sets. Fine-tuning is typically very fast, so it is reasonable to simply run an exhaustive search over the above parameters and choose the model that performs best on the development set.

3.6 Comparison of BERT and OpenAI GPT

The most comparable existing pre-training method to BERT is OpenAI GPT, which trains a left-to-right Transformer LM on a large text corpus. In fact, many of the design decisions in BERT were intentionally chosen to be as close to GPT as possible so that the two methods could be minimally compared. The core argument of this work is that the two novel pre-training tasks presented in Section 3.3 account for the majority of the empirical improvements, but we do not that there are several other differences between how BERT and GPT were trained:

- GPT is trained on the BooksCorpus (800M words); BERT is trained on the BooksCorpus (800M words) and Wikipedia (2500M 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 emeddings during pre-training.
- GPT was trained for 1M steps with a batch size of 32000 words; BERT was trained for 1M steps with a batch size of 128000 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.

To isolate the effect of these differences, we perform ablation experiments in Section 5.1 which demonstrates that the majority of the improvements are in fact coming form the new pre-training tasks.

4 Experiments

In this section, we present BERT fine-tuning results on 11 NLP tasks.

4.1 GLUE Datasets

The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) is a collection of diverse natural language understanding tasks. Most of the GLUE datasets have already existed for a number of years, but the purpose of GLUE is to (1) distribute these datasets with canonical Train, Dev, and Test splits, and (2) set up an evaluation server to mitigate issues with evaluation inconsistencies and Test set overfitting. GLUE does not distribute labels for the Test set and users must upload their predictions to the GLUE server for evaluation, with limits on the

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 86.0	RTE 61.7	**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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.5	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number of training examples. The "Average" column is slightly different we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

number of submissions. The GLUE benchmark includes the following datasets, the descriptions of which were originally summarized in Wang et al. (2018):

MNLI Multi-Genre Natural Language Inference is a large-scale, crowdsourced entailment classification task (Williams et al., 2018). Given a pair of sentences, the goal is to predict whether the second 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 (Chen et al., 2018).

QNLI Question Natural Language Inference is a version of the Stanford Question Answering Dataset (Rajpurkar et al., 2016) which has been converted to a binary classification task (Wang et al., 2018). 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 form movie reviews with human annotations of their sentiment (Socher et al., 2013).

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 (Warstadt et al., 2018).

STS-B The Semantic Textual Similarity Benchmark is a collection of sentence pairs draw from news headlines and other sources (Cer et al., 2017). 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 new sources, with human annotations for whether the sentences in the pair are semantically equivalent (Dolan and Brockett, 2005).

RTE Recognizing Textual Entailment is a binary entailment task similar to MNLI, but with mush less training data (Bentivogli et al., 2009).[5]

WNLI Winograd NLI is a small natural language inference dataset deriving from (Levesque et al., 2011). The GLUE webpages notes that there are issues with the construction of this dataset[6], and every trained system that's been submitted to GLUE has performed worse than the 65.1 baseline accuracy of predicting the majority

[5]Note that we only report single-task fine-tuning results in this paper. Multitask fine-tuning approach could potentially push the results even further. For example, we did observe substantial improvements on RTE from multi-task training with MNLI.

[6]https://gluebenchmark.com/faq

class. We therefor exclude this set out of fairness to OpenAI GPT. For our GLUE submission, we always predicted the majority class.

4.1.1 GLUE Results

To fine-tune on GLUE, we represent the input sequence or sequence pair as described in Section 3, and use the final hidden vector $C \in \mathbb{R}^H$ corresponding to the first input token ([CLS]) as the aggregate representation. This is demonstrated visually in Figure 3 (a) and (b). The only new parameters introduced during fine-tuning is a classification layer $W \in \mathbb{R}^{K \times H}$ where K is the number of labels. We compute a standard classification loss with C and W, i.e., $log(softmax(CW^T))$.

We use a batch size of 32 and 3 epochs over the data for all GLUE tasks. For each task, we ran fine-tunings with learning rates of 5e-5, 4e-5, 3e-5, and 2e-5 and selected the one that performed best on the Dev set. Additionally, for $\text{BERT}_{\text{LARGE}}$ we found that fine-tuning was sometimes unstable on small data sets (i.e., some runs would produce degenerate results), so we ran several random restarts and selected the model that performed best on the Dev set. With random restarts, we use the same pre-trained checkpoint but perform different fine-tuning data shuffling and classifier layer initialization. We not that the GLUE data set distribution does not include the Test labels, and we only make a single GLUE evaluation server submission for each $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$.

Results are presented in Table 1. Both $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ outperform all existing systems on all tasks by a substantial margin, obtaining 4.4% and 6.7% respective average accuracy improvement over the state-of-the-art. Note that $\text{BERT}_{\text{BASE}}$ and OpenAI GPT are nearly identical in terms of model architecture outside of the attention masking. For the largest and most widely reported GLUE task, MNLI, BERT obtains a 4.7% absolute accuracy improvement over the state-of-the-art. On the official GLUE leaderboard[7], $\text{BERT}_{\text{LARGE}}$ obtains a score of 80.4, compared to the top leaderboard system, OpenAI GPT, which obtains 72.8 as of the date of writing.

It is interesting to observe that $\text{BERT}_{\text{LARGE}}$ significantly outperforms $\text{BERT}_{\text{BASE}}$ across all tasks, even those with very little training data. The effect of BERT model size is explored more thoroughly in Section 5.2.

[7] https://gluebenchmark.com/leaderboard

4.2 SQuAD v1.1

The Stanford 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. For example:

- Input Question:

  Where do water droplets collide with ice crystals to form precipitation?

- Input Paragraph:

  ... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. ...

- Output Answer:

  within a cloud

This type of span prediction task is quite different from the sequence classification tasks of GLUE, but we are able to adapt BERT to run on SQuAD in a straightforward manner. Just as with GLUE, we present the input question and paragraph as a single packed sequence, with the question using the A embedding and the paragraph using the B embedding. The only new parameters learned during fine-tuning are a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$. Let the final hidden vector from BERT for the i^{th} input token be denoted as $T_i \in \mathbb{R}^H$. See Figure 3(c) for a visualization. Then, the probability of word i being the start of the answer span is computed as a dot product between T_i and S followed by a softmax over all of the words in the paragraph:

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

The same formula is used for the end of the answer span, and the maximum scoring span is used as the prediction. The training objective is the log-likelihood of the correct start and end positions.

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)	80.9	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**

Table 2: SQuAD results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

We train for 3 epochs with a learning rate of 5e-5 and a batch size of 32. At inference time, since the end prediction is not conditioned on the start, we add the constraint that the end must come after the start, but no other heuristics are used. The tokenized labeled span is aligned back to the original untokenized input for evaluation.

Results are presented in Table 2. SQuAD uses a highly rigorous testing procedure where the submitter must manually contact the SQuAD organizers to run their system on a hidden test set, so we only submitted our best system for testing. The result shown in the table is our first and only Test submission to SQuAD. We note that the top results from the SQuAD leaderboard do not have up-to-date public system descriptions available, and are allowed to use any public data when training their systems. We therefore use very modest data augmentation in our submitted system by jointly training on SQuAD and TriviaQA (Joshi et al., 2017).

Our best performing system outperforms the top leaderboard system by +1.5 F1 in ensembling and +1.3 F1 as a single system. In fact, our single BERT model outperforms the top ensemble system in terms of F1 score. If we fine-tune on only SQuAD (without TriviaQA) we lose 0.1-0.4 F1 and still outperform all existing systems by a wide margin.

4.3 Named Entity Recognition

To evaluate performance on a token tagging task, we fine-tune BERT on the CoNLI 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).

For fine-tuning, we feed the final hidden representation $T_i \in \mathbb{R}^H$ for to each token i into a classification layer over the NER label set. The predictions are not conditioned on the surrounding predictions (i.e., non-autoregressive and no CRF). To make this compatible with WordPiece tokenization, we feed each CoNLI-tokenized input word into our WordPiece tokenizer and use the hidden state corresponding to the first sub-token as input to the classifier. For example:

Jim	Hen	##son	was	a	puppet	##eer
I-PER	I-PER	X	0	0	0	X

Where no prediction is made for X. Since the WordPiece tokenization boundaries are a know part of the input, this is done for both training and test. A visual representation is also given in Figure 3 (d). A caed WordPiece model is used for NER, wheres an uncased model is used for all other tasks.

Results are presented in Table 3. $\text{BERT}_{\text{LARGE}}$ outperforms the existing SOTA, Cross-View Training with multi-task learning (Clark et al., 2018), by +0.2 on CoNLI-2003 NER Test.

4.4 SWAG

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

Given a sentence from a video captioning dataset, the task is to decide among four choices the most plausible continuation. For example:

Adapting BERT to the SWAG dataset is similar to the adaptation for GLUE. For each example, we construct four input sequences, which each contain the concatenation of the given sentence (sentence A) and a possible continuation (sentence B). The only task-specific parameters we introduce is a vector $V \in \mathbb{R}^H$, whose dot product with the final aggregate representation $C_i \in \mathbb{R}^H$

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

Table 3: CoNLL-2003 Named Entity Recognition re- sults. The hyperparameters were selected using the Dev set, and the reported Dev and Test scores are aver- aged over 5 random restarts using those hyperparame- ters.

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
Humam	-	85.0
Human (5 annotations)	-	88.0

Table 4: SWAG Dev and Test accuracies. Test results were scored against the hidden labels by the SWAG au- thors. †Human performance is measure with 100 sam- ples, as reported in the SWAG paper.

denotes a score for each choice i. The probability distribution is the softmax over the four choices:

$$P_i = \frac{e^{V \cdot C_i}}{\sum_{j=1}^{4} e^{V \cdot C_j}}$$

We fine-tune the model for 3 epochs with a learning rate of 2e-5 and a batch size of 16. Results are presented in Table 4. $\text{BERT}_{\text{LARGE}}$ outperforms the authors' baseline ESIM+ELMo system by +27.1%.

5 Ablation Studies

Although we have demonstrated extremely strong empirical results, the results presented so far have not isolated the specific contributions from each aspect of the BERT framework. In this section, we perform ablation experiments over a number of facets of BERT in order to better understand their relative importance.

5.1 Effect of Pre-training Tasks

One of our core claims is that the deep bidirectionality of BERT, which is enabled by masked LM pre-training, is the single most important improvement of BERT compared to previous word. To give evidence for this claim, we evaluate two new models which use the exact same pre-training data, fine-tuning scheme and Transformer hyperparameters as $\text{BERT}_{\text{BASE}}$:

1. **No NSP**: A model which is trained using the "masked LM" (MLM) but without the "next sentence prediction" (NSP) task.
2. **LTR & NO NSP**: A model which is trained using a Left-to-Right (LTR) LM, rather than an MLM. In this case, we predict every input word and do not apply any masking. The left-only constraint was also applied at fine-tuning, because we found it is always worse to pre-train with left-only-context and fine-tune with bidirectional context. Additionally, this model was pre-trained without the NSP task. This is directly comparable to OpenAI GPT, but using our larger training dataset, our input representation, and our fine-tuning scheme.

Results are presented in Table 5. We first examine the impact brought by the NSP task. We can see that removing NSP hurts performance significantly on QNLI, MNLI, and SQuAD. These results demonstrate that our pre-training method is critical in obtaining the strong empirical results presented previously.

Next, we evaluate the impact of training bidirectional representations by comparing "No NSP" to "LST & No NSP". The LTR model performs worse than the MLM model on all tasks, with extremely large drops on MRPC and SQuAD. For SQuAD it is intuitively clear that an LTR model will perform very poorly at span and token prediction, since the token-level. hidden states have no right-side context. For MRPC is unclear whether the poor performance is due to the small data size of the nature of the task, but we found this poor performance to be consistent across a full hyperparameters sweep with many random restarts.

In order make a good faith attempt at strengthening the LTR system, we tried adding a ran-

domly initialized BiLSTM on top of it for fine-tuning. This does significantly improve results on SQuAD, but the results are still far worse than the pre-trained bidirectional models. It also hurts performance on all four GLUE tasks.

We recognize that it would also be possible to train separate LTR and RTL models and representation token as the concatenation of the two models, as ELMo does. However: (a) this is twice as expensive as a single bidirectional model; (b) this is non-intuitive for tasks like QA, since the RTL model would not be able to condition the answer on the question; (c) this it is strictly less powerful than a deep bidirectional model, since a deep bidirectional model could choose to use either left or right context.

5.2 Effect of Model Size

In this section, we explore the effect of model size on fine-tuning task accuracy. We trained a number of BERT models with a differing number of layers, hidden units, and attention heads, while otherwise using the same hyperparameter and training procedure as described previously.

Results on selected GLUE tasks are shown in Table 6. In this table, we report the average Dev Set accuracy from 5 random restarts of fine-tuning. We can see that larger models lead to a strict accuracy improvement across all four datasets, even for MRPC which only has 3600 labeled training examples, and is substantially different from the pre-training tasks. It is also perhaps surprising that we are able to achieve such significant improvements on top of models which are already quite large relative to the existing literature. For example, the largest Transformer explored in (Vaswani et al., 2017) is (L=6, H=1024, A=16) with 100M parameters for the encoder, and the largest Transformer we have found in the literature is (L=64, H=512, A=2) with 235M parameters (Al-Rfou et al., 2018). By contrast, $\text{BERT}_{\text{BASE}}$ contains 100M parameters and $\text{BERT}_{\text{LARGE}}$ contains 340M parameters.

It has been known for many years that increasing the model size will lead to continual improvements on large-scale tasks such as machine translation and language modeling, which is demonstrated by the LM perplexity of held-out training data shown in Table 6. However, we believe that this is the first work to demonstrate that scaling to extreme model sizes also leads to large improvements on every small scale tasks, provided that the model has been sufficiently pre-trained.

5.3 Effect of Number of Training Steps

Figure 5.4 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 (128000 words/batch * 1000000 steps) to achieve high fine-tuning accuracy?
   Answer: Yes, $\text{BERT}_{\text{BASE}}$ achieves almost 1.0% additional accuracy on MNLI when trained on 1M steps compared to 500k steps.

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 BERTBASE 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 ran- domly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

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

Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of at- tention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

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.

5.4 Feature-based Approach with BERT

Layers	Dev F1
Finetune All	96.4
First Layer (Embeddings)	91.0
Second-to-Last Hidden	95.6
Last Hidden	94.9
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1
Sum All 12 Layers	95.5

Table 7: Ablation using BERT with a feature-based ap- proach on CoNLL-2003 NER. The activations from the specified layers are combined and fed into a two-layer BiLSTM, without back-propagation to BERT.

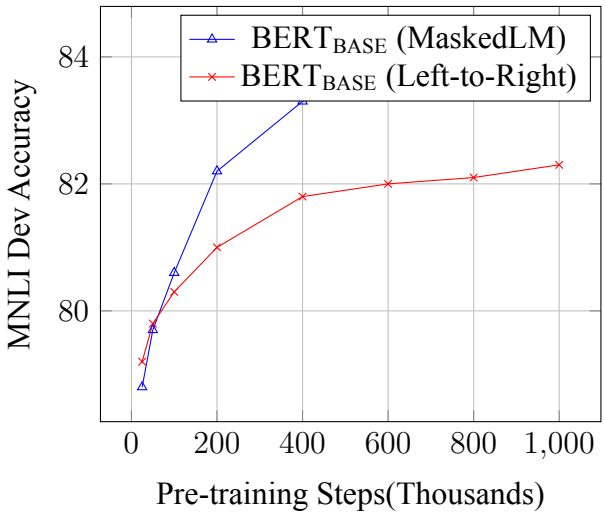

Figure 4: Ablation over number of training steps. This shows the MNLI accuracy after fine-tuning, starting from model parameters that have been pre-trained for k steps. The x-axis is the value of k.

6 Conclusion

Recent empirical improvements due to transfer learning with language models have demonstrated that rich, unsupervised pre-training is an integral part of many language understanding systems. In particular, these results enable even low-resource tasks to benefit from very deep unidirectional architectures. Our major contribution is further generalizing these findings to deep *bidirectional* architectures, allowing the same pre-trained model to successfully tackle a broad set of NLP tasks.

While the empirical results are strong, in some cases surpassing human performance, important future work is to investigate the linguistic phenomena that may or may not be captured by BERT.

References

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. *CoRR*, abs/1808.04444, 2018. URL arXivpreprintarXiv:1808.04444.

Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6, 2005. URL http://www.jmlr.org/papers/volume6/ando05a/ando05a.pdf.

Luisa Bentivogli, Bernardo Magnini, do Dagan, Hoa Trang Dang, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. *CoRR*, 2009.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspon- dence learning. *CoRR*, 2006.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, 2015.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C

Lai. Class-based n-gram models of natural language. *CoRR*, 1992.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *CoRR*, 2017.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Thorsten Brants Qi Ge, Phillipp Koehn, and Tony Robin-son. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, 2013.

Z. Chen, H. Zhang, X. Zhang, and L. Zhao. Quora question pairs. *CoRR*, 2018.

Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc V Le. Semi-supervised sequence modeling with cross-view training. *CoRR*, 2018.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *CoRR*, 2008.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Lo ̈ic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, 2017.

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. *CoRR*, 2015.

J. Deng, W. Dong, R. Socher, K. Li L.-J. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *CVPR09*, 2009.

William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. *CoRR*, 2005.

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *ACL*, 2018.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehen- sion. *ACL*, 2017.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. *CoRR*, 2015.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, 2014.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. *CoRR*, 2011.

Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. *CoRR*, 2018.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Con- textualized word vectors. *NIPS*, 2017.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Cor-rado, and Jeff Dea. Distributed representa- tions of words and phrases and their compositional- ity. *CoRR*, 2013.

Jeffrey Pennington, Richard Socher, and Christo pher D. Manning. Glove: Global vectors for word representation. *CoRR*, 2014.

Matthew Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised se- quence tagging with bidirectional language models. *ACL*, 2017.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoye. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. *CoRR*, 2018.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *CoRR*, 2016.

Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *CoRR*, 2003.

Richard Socher, Alex Perelygin, Jason Chuang Jean Wu, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment tree- bank. *CoRR*, 2013.

Wilson L Taylor. cloze procedure: A new tool for measuring readability. *CoRR*, 1953.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. *CoRR*, 2010.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, 2017.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. *CoRR*, 2008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018. URL http://arxiv.org/abs/1804.07461.

A. Warstadt, A. Singh, and S. R. Bowman. Corpus of linguistic acceptability. *CoRR*, 2018.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *NAACL*, 2018.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Mohammad Norouzi Quoc V Le, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, and et al Klaus Macherey. Google's neural ma- chine translation system: Bridging the gap between human and machine translation. *CoRR*, 2016.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neuralnetworks? *CoRR*, 2014.

Rowan Zellers, Yonatan Bisk, Roy Schwartzz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *CoRR*, 2018.

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. *CoRR*, 2015.