Transferability of Contextual Representations for Question Answering

Travis McGuire
Department of Electrical Engineering
Stanford University
`travis.mcguire@stanford.edu`

Pranav Bhardwaj
Department of Statistics
Stanford University
`pranavb@stanford.edu`

Abstract

Large pretrained natural language representations such as BERT, ALBERT, and other variants (BERT-models) have achieved state of the art performance on a diverse set of NLP tasks after fine tuning. This suggests these BERT-models learn to extract signal-rich, transferable language features. We investigate whether there is an appreciable difference in feature quality at various depths of pretrained and fine tuned BERT-models by training softmax regression probes to perform a semantic level task, question answering (QA) on SQuAD 2.0. We find that the feature quality of fine tuned BERT-models improves with each successive layer, while the feature quality of pretrained BERT-models remains constant. We also find that pretrained BERT-models feature quality is relatively poor in comparison to high layers in the fine tuned models, suggesting that the fine tuning process is key for extracting high quality features. One particularly interesting finding is that the early to middle layers in fine tuned BERT-models begin to perform well on questions with answers, at the cost of performance on questions with no answer. Higher layers in fine tuned BERT-models are able to perform well on both questions with and without answers.

Code available at https://github.com/travismcguire/cs224nfinalproject

1 Introduction

BERT-models achieve incredible success on many NLP tasks [1]. However, it is difficult to understand the learning process that makes these models successful. The difficulty is perhaps best demonstrated by the over two hundred references within a recent survey on how to analyze, interpret, and evaluate neural networks for natural language processing [2]. Developing an understanding of this learning process is important, as current empirical evidence suggests most improvements are expensive – coming from more model complexity, more data, or more training time [3].

While there are many BERT-models to understand, ALBERT-base-v2 is presented in this work based on its state of the art performance [4]. Henceforth ALBERT-base-v2 is referred to as ALBERT, but the reader should note there are larger ALBERT architectures with more and larger layers. The hope is these results are applicable to BERT's other variants and larger ALBERT architectures, but the authors would be excited to see evidence which supports or contradicts that hope. As a baseline and point of comparison, results using the original BERT architecture are also presented [1].

This effort to improve understanding relies on probing. Softmax regression probes are trained and evaluated on the question answering (QA) language task using the attention-weighted hidden states (extracted features) of various BERT-models as inputs.

ALBERT and BERT use 12 encoders (often called hidden layers, or feature extractors) in a hierarchical fashion. This work fits separate softmax regression probes for each layer. This will provide insights into how the hidden representations, and their utility with respect to question answering, evolve with depth. Some have hypothesized lower layers provide more syntactic information (e.g. is `people` a noun?) while higher layers provide more semantic information (e.g. answering `where do people come from?`). A number of authors have looked at the syntactic understanding of lower BERT-model layers [5][6][7][8][9][10][11]. Semantic understanding of BERT-model layers has been investigated less than syntactic understanding, but has been touched on by several works [5][7][11][8][12].

Following the BERT-model paradigm, ALBERT and BERT are pretrained for a generic language modeling task, then fine tuned for specific language task [4][13]. Pretrained models, as expected, perform worse than fine tuned ones since they were not trained specifically for the task, but are the underlying features extracted different [14]? This work presents probing results for both pretrained and fine tuned ALBERT and BERT, allowing the reader to consider the differences in features extracted between them.

2 Related Work

BERT, standing for Bidirectional Encoder Representations from Transformers, is built out of blocks of a neural architecture known as transformers [1][15]. Transformers are based on attention, see the following for detailed explanations of these processes [16][17][18]. The original BERT model is pretrained on two large text corpora (Wikipedia and BookCorpus) for masked word prediction and next sentence prediction. It is adapted to predict answers for QA on SQuAD 2.0 using a softmax regression architecture, which served as inspiration for the softmax regression probes used in this work [1]. ALBERT is a variant of BERT with reduced parameters through projecting the large vocabulary matrix to a smaller subspace and cross-layer parameter sharing [4]. ALBERT-base-v2 is pretrained on large amounts of unstructured data for a sentence ordering task (i.e. given two segments of text, which should come first) [19]. ALBERT is then fine tuned for a specific language task by adding an appropriate classifier layer and propagating gradients through the entire network as seen with the original BERT architecture. A key difference to note in this work is that gradients are only propagated to the softmax regression probes, while the state of ALBERT and BERT are kept frozen.

The original BERT paper investigates using BERT as a feature extractor and training only the softmax regression classifier for named entity recognition its ablation studies, and finds BERT is effective for feature-based approaches [1]. The original BERT paper [1] has more detail. Subsequent research in the same spirit, Bertology, has grown rapidly [13].

The authors of ELMo, another contextual word representation architecture which uses a bi-directional LSTM instead of transformers, found evidence that the lower layer provides more syntactic features while its higher layer provides more semantic features [5][20]. One work, highly inspirational to this one, follows up by looking at performance for various layers of both ELMo and BERT on a variety of language tasks [6]. They support the prior findings for ELMo (bi-directional LSTM) by seeing the lower layers are most transferable. They also find that the middle layers of BERT (bi-directional transformer) are most transferable for a range of mostly syntactic tasks. One group comparing contextual architectures like BERT and ELMo to prior non-pretrained contextual baseline found that the contextual architectures have an advantage in syntactic tasks that is lessened as one shifts to more semantic tasks [7]. One of the same authors in a following work inspects BERT layer-by-layer, and found the features encoded parallel the hierarchical relations in a traditional NLP pipeline (with some caveats) [8].

Alternative probing methods have been proposed, such as using structural probes to provide evidence of whether the model has learned syntactic trees (a tree showing the structure of the text according to some grammar) [9]. Syntactic tree learning has been investigated using information from the attention heads instead of probing entirely [10]. Others have used attention for investigations, seeing what attention does for syntax and coreference tasks (coreference is a semantic task, specifically to find which entity words like `it` and `they` are referencing) [11]. As an alternative to language tasks entirely, one work looks at how these pretrained contextual embedding models like ELMo and BERT, along with more tradition approaches, handle numerical tasks [12]. They show that although the models can do some math, they fall short of their non-pretrained contextual counterparts.

3 Approach

The task addressed in this work is question answering (QA). Given a natural language question and context paragraph, QA systems are tasked with answering the question, or stating that the context does not provide sufficient information to answer the question. QA has many practical applications, such as providing answers to questions in confusing legal documents or long documentation. The authors consider question answering to be a high level semantic task, which requires nuanced understanding of the relationship between the question, context, and answer. Below the QA system pipeline is detailed. This pipeline takes as input a question and answer in natural language. This natural language is tokenized. A BERT-model is then used to extract features from this tokenized input. Softmax regression probes utilize these features to make predictions, and finally these predictions are turned into a natural language answer.

3.1 Natural Language to Tokens

The first thing that needs to be done is represent the question and answer in a mathematically useful way. Therein lies the importance of a tokenizer. Tokenizers convert strings into a sequence of tokens. One could imagine a simple example, would be splitting on spaces to make each word its own token. Another option is splitting on each entry to make each character its own token. One can imagine a whole cottage industry around tokenizers, doing clever things like splitting the word $['transformed']$ into two tokens $['transform', 'ed']$. The vocabulary size must generally be limited for computational and complexity reasons (e.g. 30000 important tokens), and tokenizers can address this with use of an unknown, or $[UNK]$, token.

Looking at QA specifically, one will quickly note we have two different strings. A question like `how tall is lebron james`, and a context paragraph like `LeBron's height is more luck than genes. 1 in every 15,737 18-year-olds is 6'8"`.[1]

The tokenizer will combine these two strings to create a single vector

$$h_0 = [[CLS], [Question], [SEP], [Context], [PAD], [SEP]] \in \mathbb{Z}^{\text{max seq len}} \tag{1}$$

which is always padded or truncated to be of size $\mathbb{Z}^{\text{max seq len}}$. Note that $\mathbb{Z}$ is used to indicate each entry can take a discrete value from 1 to the token vocabulary size.

3.2 Tokens to Hidden States

There are many great resources, in addition to the original papers and those mentioned in the related works section, explaining BERT-model architectures[2] and the transformer blocks of which they are composed[3]. Each encoder, also called a hidden layer, of the model takes in the previous encoder's output as an input. The an attention weighted sum of the input is fed to a fully connected network to produce the encoder's own hidden state output.

$$h_i = \text{Encoder}_i(h_{i-1}) \in \mathbb{R}^{\text{max seq len} \times \text{hidden size}} \text{ for } i = 1...n_{layers} \tag{2}$$

The input of the first encoder, which converts tokens to hidden states, is the token vector of size $\mathbb{Z}^{\text{max seq len}}$. The input of each encoder after the first is of size $\mathbb{R}^{\text{max seq len} \times \text{hidden size}}$. Note the transition to $\mathbb{R}$, as the output of the fully connected layers are real valued instead of discrete.

3.3 Hidden States to Prediction

At each hidden layer, the output h_i is extracted from the BERT-models and used as input to a probe. Each probe consists of a start network S, which predicts the start index of the answer within the context, and an end network E, which predicts the end index of the answer. The start and end index prediction networks each contain a weight vector $w \in \mathbb{R}^{\text{hidden size}}$ and a bias vector $b \in \mathbb{R}$. Questions with no answer are considered to span the start token $[CLS]$.

Predictions are made via a softmax of the form

$$\hat{y} = \sigma(h_{layer} * w + b) \tag{3}$$

where $\sigma(\cdot)$ is a vector softmax applied for a probability distribution across the *(max seq len)* axis.

[1]https://www.quora.com/How-did-LeBron-James-grow-so-tall
[2]http://jalammar.github.io/illustrated-bert/
[3]https://jalammar.github.io/illustrated-transformer/

The prediction score for an answer is calculated such that $\hat{y}_S[j] + \hat{y}_E[k]$ is maximized subject to $j \leq k$ for $j, k \neq 0$. The prediction score for no answer is calculated as $\hat{y}_S[0] + \hat{y}_E[0]$. The j, k for the maximum score are chosen, where $j, k \in 1$...*(max seq len)* represent the start and end index of the answer within the context paragraph.

3.4 Prediction to Natural Language Answer

Once indices j, k are predicted by the probes, they are used to slice into the tokenized context paragraph, which is then de-tokenized by applying the same tokenizer process in reverse to obtain a natural language answer to the question. The natural language answer from the model will be referenced as $\hat{a}$, while the ground truth answer will be referenced as a.

4 Experiments

4.1 Data

The models are trained using the SQuAD 2.0 dataset provided by Stanford University's CS224N: Natural Language Processing with Deep Learning [21]. The train set is identical to the official SQuAD 2.0 train set, the dev set contains half of the official SQuAD 2.0 dev set, and the test set consists of the other half of the official SQuAD 2.0 dev set along with some examples hand labeled by the course staff. The data consists of (context, question, answer) triples.

Split	**Size**
Train	129941
Development	6078
Test	5915

Of these triples, around 1/3 have no answer in training and 1/2 have no answer in dev and test. Having no answer as a possibility expands on a weakness of SQuAD 1.0, where it was guaranteed a question's answer was in the span of the context paragraph. SQuAD 2.0 still does have some limitations, as there are only span-based answers, questions were formed from the passages, and there are very few questions which require multiple facts spread across multiple sentences. SQuAD has multiple golden (true) answers, allowing for some flexibility in the prediction (e.g. 10th century and `the 10th century` could both be considered true). For more details, see the original SQuAD 2.0 paper [21].

4.2 Evaluation Method

Performance on SQuAD is typically measured by two metrics exact match (EM) and F1 score (F1). EM is binary, with label true if the model's predicted answer exactly matches the ground truth. F1 is the harmonic mean of precision and recall between the predicted answer and ground truth. The final leaderboard score is computed with EM and F1 with some caveats described in the CS224N handout [22]. EM is the slightly harsher of the two metrics, requiring an exact match to a golden answer.

$$EM = \sum_{l=1}^{n_{examples}} \frac{1\{\hat{a}_l = a_l\}}{n_{examples}} \tag{4}$$

$$F1 = 2 \times \frac{\text{precision} + \text{recall}}{\text{precision} \times \text{recall}} \tag{5}$$

Qualitative error analysis will also be conducted in the analysis section to help understand what types of errors are made and how they vary across the probes.

4.3 Experimental Details

4.3.1 Fine Tuning

ALBERT-base-v2 and BERT-base-uncased were fine tuned on the SQuAD 2.0 dataset for 3 epochs with a batch size of 8, maximum gradient norm clipping of 1.0, learning rate of $3e-5$, maximum sequence length of 384, and AdamW optimizer [23]. All fine tuned models have been open sourced on the transformers library community page, and have already been downloaded over a thousand times by researchers [24].

4.3.2 Probe training

Softmax regression probes were all trained in the same manner to ensure comparability. Weights were initialized with a truncated normal distribution $\mathcal{N}(0, 0.2)$, while bias was initialized to be 0. Probes were trained in a way directly paralleling the model fine tuning process – on the SQuAD 2.0 dataset for 3 epochs with a batch size of 8, maximum gradient norm clipping of 1.0, learning rate of $3e-5$, maximum sequence length of 384, and AdamW optimizer [23]. Gradients were not propagated through the entire system while training probes. Instead the BERT-models were frozen and only the probes themselves were trained, thus the BERT-models were used as feature extractors.

4.3.3 Probe prediction

The original BERT and ALBERT authors tune a threshold, a large positive constant added to the null prediction score (i.e. biasing the model to predict no answer), to maximize performance (F1 score) on the development set. After doing an initial search, we noticed the gains from such tuning to be relatively small before causing significant performance losses. Thus, we opted to not add this additional hyperparameter to tune for simplicity, and because we believe that using no threshold for each probe seems to allow comparison more readily.

4.4 Results

Figure-1 and Figure-2 show the performance of these probes on unseen dev set examples. We observe that probes utilizing pretrained representations perform similarly in all layers. These probes mostly predict no answer. In the probes utilizing fine tuned representations, we see performance increase with layer, suggesting that feature quality increases after each transformer block.

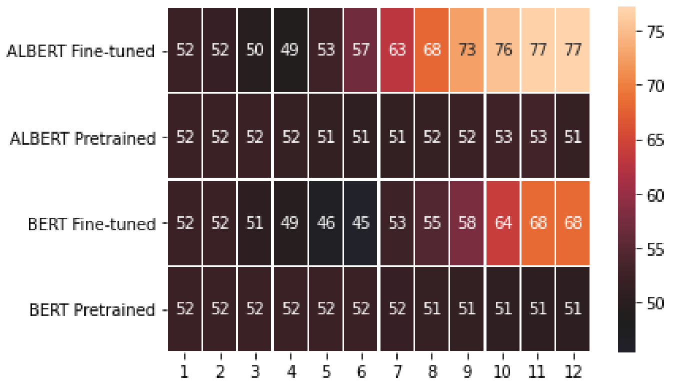

Figure 1: Probe EM by Model Layer

5 Analysis

We find that probes utilizing pretrained BERT-models, as well as probes utilizing lower layers of fine tuned BERT-models, often predict no answer. This is not merely a biasing problem, as when a threshold is used to force probes for these layers to predict answers the gains in F1 score for samples with answers are extremely slim. Indeed, in the fine tuned BERT-models we find lower layers achieve

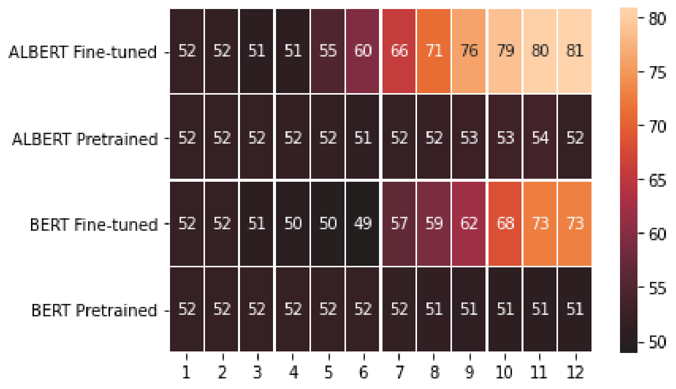

Figure 2: Probe F1 by Model Layer

Model and Layer	Example Prediction
ALBERT Fine-tuned L3	"" (no answer)
ALBERT Fine-tuned L6	"" (no answer)
ALBERT Fine-tuned L9	"traveling salesman problem and the integer factorization problem"
ALBERT Fine-tuned L12	"computational problem"
BERT Fine-tuned L3	"" (no answer)
BERT Fine-tuned L6	"the traveling salesman problem"
BERT Fine-tuned L9	"integer factorization problem"
BERT Fine-tuned L12	"computational problem where a single output (of a total function)..."
Ground truth	"computational problem"

Table 1: Predictions on question id f9b1c7ca8e161ade4d1cf4fb1

very high EM on no answer questions (because they are predicting all no answer), and higher layers gradually reduce EM on no answer questions while correctly answering questions.

Going from low to middle layers in a fine tuned BERT-model, one can see the no answer F1 decrease at a rate around proportional to the increase in answer f1 (full F1 scores, with split on answer and no answer type are included in the linked Github). Then, from middle to high layers in a fine tuned BERT-model, the F1 score for both answer and no answer increase in tandem to provide an overall improvement in F1 score.

This contrasts the case of a pretrained model, where the layers only ever provide features that lead to no answer prediction. This may indicate when fine tuning that the early to middle layers learn features useful to predicting answer indices to questions with answers, while the later layers create continually nuanced features which allow both better has answer prediction and answer/no answer discernment. This is supported anecdotally in some inspected examples 1. Pretrained model examples are not included in the table, as at each of the layers they have predicted no answer.

6 Conclusion

We find evidence supporting the importance of fine tuning of pretrained natural language representations to extract features useful for semantic level tasks, particularly in higher layers. We additionally find that pretrained natural language representations are not immediately useful for semantic level tasks. The early to middle hidden layers of a fine tuned BERT-model learn to predict answers at the cost to performance in predicting on questions without answers, while in middle to high layers the features encode information which improves performance on questions with and without answers. The primary contributions of this work are the code necessary to train, predict with, and evaluate the probes at each layer of the popular ALBERT and BERT model architectures – along with interpretation of those results. This would not have been as feasible or easy without the great utility provided by Huggingface's transformers library [25], where we have also open sourced all of the question answering models we trained. At the time of this submission, we have also fine tuned a 6 layer DistilBERT-base model and are training probes for further comparison and possible insights into the distilation process.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[2] Yonatan Belinkov and James Glass. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72, 2019.

[3] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

[4] Google Research. Albert: A lite bert for self-supervised learning of language representations. `https://github.com/google-research/ALBERT`, 2020.

[5] Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[6] Nelson F Liu, Matt Gardner, Yonatan Belinkov, Matthew Peters, and Noah A Smith. Linguistic knowledge and transferability of contextual representations. *arXiv preprint arXiv:1903.08855*, 2019.

[7] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*, 2019.

[8] Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. *CoRR*, abs/1905.05950, 2019.

[9] John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, 2019.

[10] Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R Bowman. Do attention heads in bert track syntactic dependencies? *arXiv preprint arXiv:1911.12246*, 2019.

[11] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of bert's attention. *CoRR*, abs/1906.04341, 2019.

[12] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? probing numeracy in embeddings. *arXiv preprint arXiv:1909.07940*, 2019.

[13] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *arXiv preprint arXiv:2002.12327*, 2020.

[14] Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4365–4374, Hong Kong, China, November 2019. Association for Computational Linguistics.

[15] Jay Alammar. The illustrated bert, elmo, and co. `http://jalammar.github.io/illustrated-bert/`, 2018.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[17] Alexander Rush. The annotated transformer. https://nlp.seas.harvard.edu/2018/04/03/attention.html, 2018.

[18] Jay Alammar. The illustrated transformer. http://jalammar.github.io/illustrated-transformer/, 2018.

[19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2020.

[20] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[21] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

[22] Stanford CS224N. Cs 224n default final project: Question answering on squad 2.0. http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf, 2020.

[23] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[24] Travis McGuire and Pranav Bhardwaj. Albert-base v2 trained on squad 2.0. https://huggingface.co/models?search=twmkn9, 2020.

[25] Thomas Wolf, L Debut, V Sanh, J Chaumond, C Delangue, A Moi, P Cistac, T Rault, R Louf, M Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv, abs/1910.03771*, 2019.