The Effect of Embeddings on SQuAD v2.0

Luis A. Perez [*]
Department of Computer Science
Stanford Univeristy
Stanford, CA 94305
`luis0@stanford.edu`

Abstract

In this paper, we explore several extensions of BERT for the Stanford Question and Answer v2.0 task in NLP, and summarize our analyzes of these results. In particular, we explore the benefits of using a fined-tuned BERT model for word-embdeddings underlying different, more complex question-and-answer architectures.

We continue by exploring different more traditional QA architectures, such as a BiDAF and a few other customized models. Additionally, we experiment with replacing the BERT embeddings with GPT-2 [6] embeddings. We find that model complexity provides smaller benefit to performance, compared to improved embeddings. We also find that fine-tuning for longer epochs leads to higher performance. We conclude by submitting our single-best performing model achieving a test F1 75.449, EM 72.206 on never-before seen held-out test-set.

1 Introduction

Word embeddings have shown to be an effective at improving many NLP tasks, such as sentence classification and question-and-answer [2]. The effect of these language embeddings is pronounced on performance, and word-level embeddings such as word2vect [3] and GloVe [4] have been prevelant across the NLP and NMT community for years. Recent advances in pre-training on large unspervised corpus of data have led to much improved, context-aware embeddings, such as ELMO [5], BERT [1], and GPT-1 and GPT-2 [6]. While these new networks are typically introduced and trained on the task of language modeling, the output states can be thought of as context-aware word-embeddings, and used similarly to those of word2vec and Glove. This is one of the main insights presented in this paper, and carried through the rest of the work to motivate our experiments.

It has become clear in the NLP community that given sufficient data, it's possible to achieve close to human-level performance on many tasks. However, this level of performance in only possible on the few tasks which have millions of training examples. The goal of pre-training, as introduced in ELMO and BERT< is to exploit large corpus of data to gather underlying signals from word-represetations. Similar to how word2vec captures some level of "closeness" between words, by being trained on the context-prection task, pre-training tasks attempt to capture the semantic meaning of words rather than any task-specific skill. While these models are nonetheless trained on specific tasks (such as masked language modeling or predicting the next word), these tasks are though to be generic, and the expectation is that common and important word-features will be learned by the model.

Effectively, a general strategy has emerged in NLP tasks that involves pre-training on a large corpus of untrained data (or, more commonly, utilized a pre-trained model) and then simply *fine-tuning* a far simpler model on top of the larger model. This work can be thought of as having begun with word2vec and other "word-embeddings", and have progressed at each iteration into more complex embeddings. Existing state-of the art work such as GPT-2 [6] is tasked with generating not only an embedding

[*]

for a single-word, but an embedding which depends on the surrounding context. The model size has reached over 1.5B parameters (the largest, and unreleased, GPT-2 model). This paper proposes that this model (and other) along with its parameters essentially constitute a simple embedding function $e = f_\theta(c)$ where c is our context for some interesting word and $e \in \mathbb{R}^d$ is an embedding into an d dimensional space of our choice. Whether these embeddings are pre-computed for all words (such as word2vec) or must be generated during training/inference due to context changes (as in ELMO, BERT, GPT), is not of material significance.

However, a systematic analysis of the effect of these embeddings compared to model complexity (on top of the embeddings themselves) has not been carried out. Having a clear understanding of the trade-offs here can help guide future research and industrial efforts. In this paper, we focus on a single task, the Stanford Question and Answer Task (v2.0), thereafter referred to as SQuAD v2.0, and investigate the performance as we swap out different sets of embeddings as well as try different models (from simply linear layer to a BiDAF [7] model). We measure performance using three metrics detailed later, consisting of F1, EM, and AvNA scores.

We conclude with a few interesting findings, after carrying out multiple training experiments. To summarize, we find that (1) improved model-embeddings lead to much larger improvements scores compared to more complex architectures and (2) increased fine-tuning leads to improved scores, with a cap. In the end, our most successful model consists of a BiDAF additional layer trained using GPT-2 embeddings fine-tuned over 6 epochs and achieve an F1 Score of 75.449, EM Score of 72.206.

2 Related Work

There is a long history of pre-training for languamge modeling. We briefly discuss some of the previous achievements.

2.1 word2vec

Learning a representation of words was a seminal task when first tackling NLP problems. The now famous word2vec representation tackled the task of using a skip-gram model of language. Given a sequence of $w_1, w_2, w_3, \cdots w_T$, the objective of the Skip-gram model is to maximize the average log-probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j|w_t})$$

where the probability is defined as the simple softmax function:

$$p(w_O \mid w_I) = \frac{\exp(v_{w_O}^{T'} v_{w_I})}{\sum_{w=1}^{W}\exp(v_w^{T'} v_{w_T})}$$

With the relatively simple model above, especially when compared with more complex modern models (such as BERT) good performance was nonetheless achieved, and SOTA results occurred for the time. The above, along with a few other techniques such as negative sampling and tricks for dealing with infrequent words were introduced in [3]. It was not long before word2vec became the de-factor embedding for many NLP tasks, such as GLUE and other datasets (SQuAD). Even now, it still remains a popular if somewhat outdated embedding technique which can often times deal relatively good results. In fact, techniques such as BERT, ELMO, and GPT continue to build on these embeddigs (and GLOVE).

2.2 GloVe

The next big breakthrough in language embeddings came about when GLOVE [4] was introduced. GLOVE combined global matrix factorization and local context window methods, two big advantages of existing systems. While it increased the complexity of the embeddings model, it reached new SOTA performance on a few NLP tasks. However, nonetheless GLOVE remained relatively similar to the word2vec model, in that it simply added some complexity to the model for embedding individual words into static feature spaces. The main difference that needs to be emphasized for these embedding models is that for a single word w_i, the embedding was fixed.

2.3 ELMO

Further advancements continued, bringing along increasingly complex models. However, the biggest revolution occured with the work of ELMO, which introduced new context-aware embeddings. In a way, ELMO presented itself no longer as word-embeddings, but instead as network pre-training. This is similar to the network pre-training that was popular in the image recognition community at the time. In fact, ELMO can produce drastically different representations for the same word, since in fact, it is dependent on the context [5].

Another departure for ELMO is that the model itself because extremely deep – it was, in fact, based on a bi-directional LSTM model with multiple level of depth (see Figure 2.3. This increased the model capacity significantly, leading to embeddings which produced SOTA results across the board, by significant margins. Another small departure was due to the fact that the representations for a given word were entirely based on on the characters.

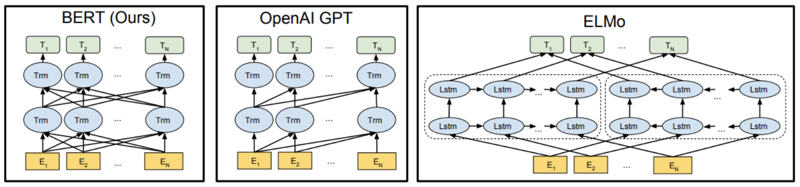

Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations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

2.4 BERT and GPT-1 and GPT-2

The final and most recent advancements relevant to this work involve the introduction of transformer architectures in the pre-training (or embedding) model. Attention-only models, introduced in [8], had led to improved performance from a model-comparison perspective in multiple tasks. However, this increased performance, while delivering SOTA results, was nonetheless still smaller than the switch to larger and more complex models.

As for GPT-1 and GPT-2, there's very little difference between the two [6] other than increasing the model size (XB parameters) and increasing the amount of data on which they're trained.

3 Approach

We begin by setting a baseline system where BERT is used with a single additional layer, fine-tuned on the SQuAD dataset as presented in the original paper [1] for 2 epochs. We demonstrate that such a system achieves great performance, as measures by F1, EM, and Answer-vs-No-Answer (AvNA) metrics.

In the paper by Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova from the Google AI Language Lab titled "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [1], BERT, an LM, is shown to have great promise in a wide variety of tasks. In particular, it perform extremenly well on the SQuAD v2.0 challenge. See Table 1 for a list of the top-ten models, and note that all of these models make use of the BERT architecture + weights as a foundation for their work.

BERT is a pre-trained, non-task-specific language model which has been used to further the SOTA results in task-specific metrics, such as GLUE and SQuAD v1.1. Other similar language models exists, such as ELMO [5] and GPT-2 [6], each making use of a large corpus of language traning data in order to generate *deep contextualized* word representations, rather than static, fixed-size word representations used previously, such as those in word2vec. These representations can better model complex characteristics of word use and how these uses vary across lingustic contexts. BERT, specifically, follows the same core idea as ELMO – extending the embeddings for a word be using

Model/Method	Category	EM	F1
BERT + MMFT+ ADA	BERT Ensemble	85.082	87.615
BERT + Synthetic Self-Training	BERT Ensemble	84.292	86.967
BERT finetune baseline	BERT Ensemble	83.536	86.096
Lunet + Verifier + BERT	BERT Ensemble	83.469	86.043
PAML + BERT	BERT Ensemble	83.467	86.035
Lunet + Verifier + BERT	BERT Extension	82.995	86.035
BERT + MMFT + ADA	BERT Extension	83.040	85.892
BERT + Synthetic Self-Training	BERT Extension	82.975	85.810
PAML + BERT	BERT Extension	82.577	85.810

Table 1: Table with Top 10 models/methods on SQuAD 2.0 as of 2/12/2019. See SQuAD Leaderboard for more details.

the context in which it appears to modify them. All of these large language models vary mostly in the architecture of the neural networks.

With these pre-trained language representations, task-specific models can be utilized downstream through two main strategies (1) feature-based and (2) fine-tuning. The former essentially includes the pre-trained language representations along with additional features, to train a new model, while the latter simply fine-tunes the pre-trained parameters (without introducing too many new parameters).

3.1 Detailed Architecture

The BERT paper [1] specifically tries to solve the problem of learning an improved language model for use in word-representations. In this paper, we take a similar approach and extend it. We use the pre-trained version of the BERT weights, which have been ported to PyTorch [2]. We provide a brief overview of the architecture and the pre-training tasks.

The BERT embeddings are context-aware and are contructed from two tasks.

- The “masked language model”.
- The “next sentence prediction”.

With these pre-training objectives independent of final objectives, (2) utilization of bi-directional models to take full advantage of the language model, and (3) a pre-trained model which achieves SOTA results across multiple, distinct NLP tasks after fine-tuning, with a single additional layer. The key achievement is a 4.4% to 6.7% average accuracy improvement over previous SOTA models in the GLUE benchmarks. The full results (for all benchmarks) are reproduced in Table 3.1 from the paper. The best performing BERT system also outperforms the top leaderboard system by +1.5 F1 in ensembling and +1.3 F1 on a single system. For more details, see Table 3.1. In this paper, we use these original values as our baselines.

BERT’s model architecture is simply a multi-layer bidrectional Transformer encoder based on the original implementation by Vaswani et al [8] and as detailed in Figure 2.3. Transformers help reduce the number of operations required to learn long-distance dependencies to a constant number (at the cost of resolution). They make use of stacked self-attention and point-wise, FC layers for the encoder and decoder, as can be seen in Figure 2. A detailed discussion of this architecture (by annotating the paper that introduced it), can be found here.

This model is then trained on the BooksCorpus (800M words) and English Wikipedia (2,500M words) (only text, no lists/tables/headers). The model we use is trained on the MLM task. In this context, the input data has tokens randomly masked (removed), and the goal is to generate a model that, given this incomplete data, can correctly identify the original vocabulary id. One key aspect of this task is that the model has access to both previous words **and** future words. Further, it is different from auto-regressive models in that only the hidden state for the masked-word is used to compute the softmax probabilities over the vocabulary. We defer further details in this task to [1], but suffice it to say that the learned, context-aware embeddings are quite effective at capturing the most important properties of language.

[2]PyTorch Open-Source Port of Multiple Models

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				
BERT$_{\text{BASE}}$ (Single)	80.8	88.5	-	-
BERT$_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
BERT$_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
BERT$_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
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.

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	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
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 below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); BERT$_{\text{BASE}}$ = (L=12, H=768, A=12); 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/`.

3.1.1 Our Modifications

All of our code for the experiments is publically available here. The paper explores two main objectives, all of which rely on pre-trained contextualized embeddings:

1. Network architecture comparison for fine-tuning [single-layer model, BiDAF, QANet] of BERT word-embeddings.
2. Contextualized word-embedding effect [ELMo vs BERT vs GPT-1 vs GPT-2] on most promising architecture.
3. Hyperparameter tuning during training using Bayesian Learning (see Spearmint Package) on BERT + BiDAF

3.1.2 Network Architecture Comparisons

With the network architecture comparison, we see to understand the effect of further network architectures on-top of pre-trained contextual word-embeddings. In particular, this paper answer the following questions.

- Do we need fine-tuning layer? Here, we propose an extreme, where we actually seek to analyze and understand the results of using BERT with no additional fine-tuning layers.
- What if it were deeper? Here, we propose an extension, where additional intermediate convolutional layers are used rather than a single FC layer on the classification hidden state. We suspect that this will lead to some improved performance on the fine-tuned tasks, since the model will have increased flexibility and capacity.

- Can we make use of the other transformer hidden states? Our hypothesis is that currently fine-tuned BERT models are bottlenecked by the single hidden-state. Instead, we propose making use of all of the transformer hidden states for classification

The key contribution in our paper is mostly experimental. We begin by using the BERT word-embeddings with full-network fine-tunining to perform question-and-answering. Our baseline model consists of exactly this model. We have already trained this model [3].

3.1.3 Linear Model

For this system, we take advantage of the input representation used by BERT. The question tokens correspond to the sentence 1 encoding, and the paragraph to the sentence 2 encoding. Then each output transform state, T_i is dot-produced with two new parameter vectors, S and E for start and end. The a straightforward softmax is taken from this dot-product for all i, to determine the start and end of the corresponding answer in the text. Similar modifications are used for other tasks, which are detailed further in the paper.

The metric would be to use BLEU against the correctly extracted sentence text. In a failure mode, the model would actually learn simply to find the answer text and generate it. However, we'd like to avoid this happening.

3.1.4 BiDAF Model

For this system, we compare the performance of BERT embeddings using a more complex model on top. In particular, we make use of the BiDAF model which is modified slightly to handle embeddings from different networks (BERT, ELMO, GPT-1, GPT-2).

We provide a brief overview of the architecture, but for details, see [7]. The BiDAF architecture can be best viewed in Figure 1. The basic idea of the BiDAF model is to provide a mechanims for attention to flow into the modeling layers after the contextual embedding of the words. In our experiments, we replace the contextual embeddings with five (+ baseline) possible embedding mechanisms (note that we never make use of the character-level embeddings directly, except for ELMO which makes use of them implicitly):

- word2vec - the "contextual embedding" layer consists of the bi-directional LSTM is completely replaced by a simple lookup layer into word2vec embeddings
- GLOVE - the "context embedding" layer is essentially removed so that the hidden states consists entirely of the GLOVE embeddings of the corresponding input token.
- baseline - the normal BiDAF model is used with GLOVE word-embeddings.
- ELMO - the contextual embed layer is replaced by a pre-trained ELMO network (also replaces the word-embed layer, makes use of character tokens)
- BERT - the contextual embed layers is replaced by the a pre-trained BERT network (also replaces the word-embed layers, does not make use of character tokens)
- GPT-2 - the contextual embed layer is replaced by a pre-trained GPT-2 network (also replaces word-embed layer, does not make use of character tokens)

4 Experiments and Analysis

The best-performing experimental results have been submitted to the PCE-division of the DEV board and the TEST board. They consist of a fully-trained BiDAF model making use of GPT-2 word-embeddings for the contextual embedding layer. We present the results of

4.1 How Far Do Embeddings Go?

For this set of experiments, we focus on the effect of using different contextualized embeddings. In an effort to make the experiments fair, we use the same set of hyper-parameters across all experiments

[3] code

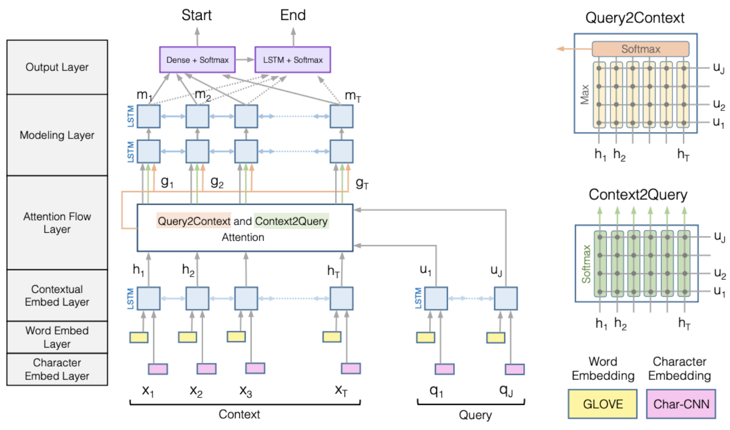

Figure 1: BiDirectional Attention Flow Model *(best viewed in color)*

Figure 1: General architecture for bideractional attention flow model. Context and query inputs are the outputs of our embedding networks (BERT, Glove, etc.) and generally replace the character/word-embed layers above. The contextual, attention flow, and modeling layers remain unmodified from the original implementation.

(learning rate of 3×10^{-5}, maximum sequence length of 384), a document stride of 128, and a train batch size of 48 sequences (context, question, answer) tuples. We use 2 gradient accumulation steps, and train over 4 GPUs (Tesla K80). This means that each batch-size on each GPU is just 6 – this was done in order to fit the data (especially for larger models) in the 8GBs of memory. We compare the same simple-linear model across 4 different embeddings. The simple model consists of two weight matrices for the start and end (see the BERT paper for details [1]) which are multiplied with each word-embedding in order to output a logit as to whether the word is the start/end of the answer query. No-anwer queries are handled by start/end both being the zero-index, which corresponds to a special no-answer token,

We summarize our results in Table 2.

Embedding Type	EM (dev)	F1 (dev)
word2vec	32.027	34.468
GLOVE	39.234	40.192
ELMO	49.991	51.291
BERT-base (baseline)	69.990	73.858
GPT-2	70.192	74.114

Table 2: EM and F1 Scores after training as specified previously across different embedding types.

4.1.1 Analysis of Results

From the above, it appears that qualitatively, the embeddings themselves actually make a significant impact on performance. With just the outlined scores as per above, we see that using a larger model that is pre-trained on a larger corpus of data gives significant advantages over simple non-context aware embeddings. It looks like the simple linear model used on-top of the embeddings is, once the semantic meaning has been extracted from the words, able to relatively easily determine the answer/no-answer scenarios (though non of the above match human level performance)

4.2 What about more complex models?

In the last set of experiments, we focus our results on the GPT-based embeddings as these were the embeddings which achieved the best results across the board. The hyper-parameters remain

Model	EM (dev)	F1 (dev)
GLOVe	63.636	67.277
BERT-base	70.114	74.752
GPT-2	72.688	76.071

Table 3: EM and F1 Scores after training as specified previously across different models.

essentially the same as previously stated. We compare GPT-based embeddings using the simple model against the BiDAF model. See Table 3 for results.

4.2.1 Analysis of Results

The above results appear to indicate that more complex models benefit simpler embeddings more, while benefitting more involved embeddings (such as BERT and GPT-2) much less. We can see this by doing an across-row comparison between the two tables for the same models.

4.3 Concluding Remarks

Overall, the paper presents an attention-based mechanism for pre-training on a large-corpus of data. Furthermore, due to the architectural choices and the input representations, this pre-trained model called BERT can be fine-tuned, with the addition of a few final layers, to achieve SOTA performance across many tasks.

We conclude that pre-trained and improved contextualized word-embeddings appear to give the biggest gains in scores on the SQuAD v2.0 metric. We dare to generalize that is is in fact, generally true. Given some pre-liminary analsis of how the network operates as presented in the previous section, we conclude that using better word-embeddings is critical to achieving good performance. Furthermore, we conclude that once a relatively good contextualized mechanism for word-level embeddings has been learned, very few additional parameters are actually nedeed to achieve good performance on a given task (in our case, SQuAD v2.0).

As such, we conclude with the following remark – future research should continue to emphasize the power of the existing transformer architecture by generating deeper and larger models which are trained on a larger corpus of data. In fact, research such as large GPT-2 by OpenAI [6] should continue, as this is likely to provide the largest benefits across many NLP tasks (and even harder tasks). Focusing on individual tasks and generating more complex but task-specific architectures for them appear, from our experiments, to provide less benefit.

In conclusion, existing large-scale transformer architectures appear to **not** have reached capacity, in terms of how well they can model languages. Researchers should continue to focus on increasing their capacity, and we expect that further gains can come from these efforts in multiple NLP tasks. On the other hand, it appears that existing architectures provide a sufficiently good and generic prior across tasks, and need not be improved further.

4.4 Future Work

It would be interesting to continue the study of larger pre-trained models. In particular, having a single model which attempts to model multiple languages might give even further advancements across many tasks.

5 Appendix

References

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] J. Howard and S. Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[4] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

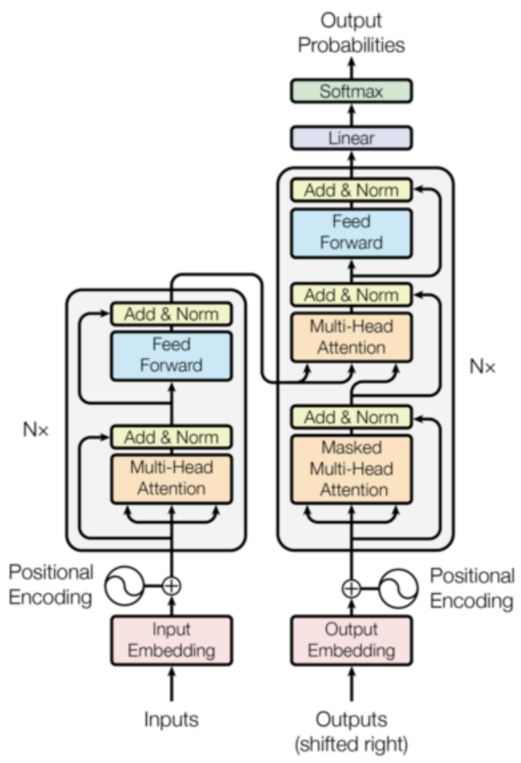

Figure 2: Transformer Architecture

[5] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[7] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.