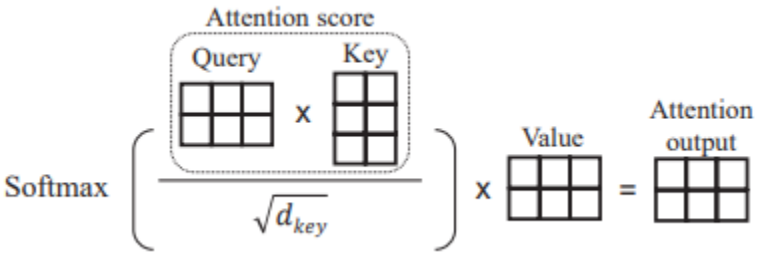
BERT and XLNet for sentiment analysis: A comparative study

A brief introduction to BERT

Bert stands for **B**idirectional **E**ncoder **R**epresentation from **T**ransformers. Introduced in 2019 [1], Bert first laid out the possibility of pre-training the Bidirectional representations. The bidirectional training of transformer allows the language model to learn self-attention This contrasts with previous models which observed a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. A transformer consists two separate mechanisms – Encoder that reads the input and decoder that produces a prediction (mostly used for Machine translation tasks). Since BERT aims to generate a language model, it uses the encoder mechanisms. As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore, it is considered bidirectional. This characteristic allows the model to learn the context of a word based on complete surroundings (left and right of the word) which is why it supports self-attention.



Strategies used by BERT

With language training models, one of the biggest challenges is defining a prediction goal. Many models predict the next word in a sequence which is a directional approach which inherently limits context learning. To overcome this challenge, BERT uses two training strategies:

Masked Language Modeling (MLM)

In MLM, 15% of the words in each sequence are replaced with a [MASK] token before feeding word sequences into BERT. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.

the prediction of the output words requires:

- Adding a classification layer on top of the encoder output.

- Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
- Calculating the probability of each word in the vocabulary with softmax.

The loss function used by BERT is Adam, which takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. Consequently, the model converges slower than directional models, a characteristic which is offset by its increased context awareness.

Next Sentence Prediction (NSP)

In the training process, the model is fed several pairs of sentences as input and it learns to predict if the second sentence in the pair is the subsequent sentence in the original document. Data input is divided 50/50 with 50% being inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus. To make the model distinguish between the two sentences, the input is processed in the following way.

- A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
- A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

To predict if the second sentence is connected to the first, the following steps are performed.

- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
- Calculating the probability of IsNextSequence with softmax .

Masked LM and Next Sentence Prediction are trained together when training BERT with the goal of minimizing the combined loss function.

Observations

- Model size matters, even at huge scale. BERT_large, with 345 million parameters, is the largest model of its kind. It is demonstrably superior on small-scale tasks to BERT_base, which uses the same architecture with “only” 110 million parameters.
- With enough training data, more training steps == higher accuracy. For instance, on the MNLI task, the BERT_base accuracy improves by 1.0% when trained on 1M steps (128,000 words batch size) compared to 500K steps with the same batch size.

- BERT's bidirectional approach (MLM) converges slower than left-to-right approaches (because only 15% of words are predicted in each batch) but bidirectional training still outperforms left-to-right training after a small number of pre-training steps.

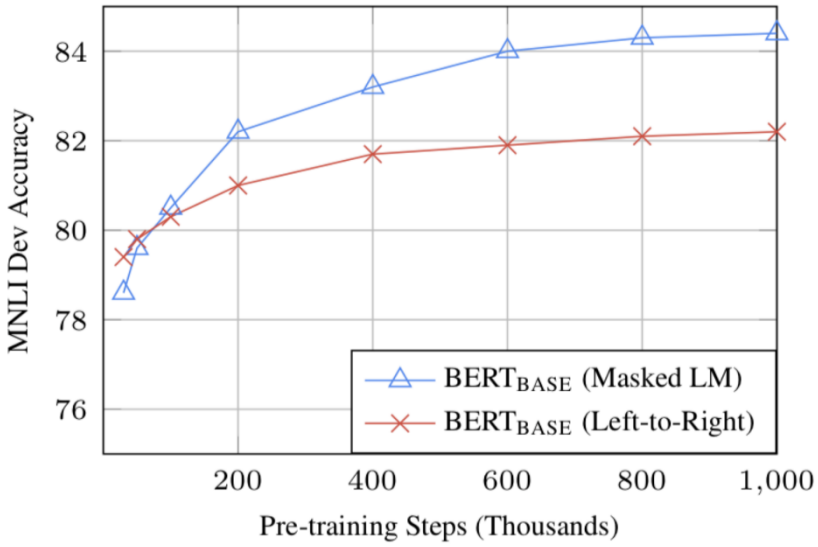


Figure 1 Source: BERT [Devlin et al., 2019]

XLNET

XLNet is a generalized autoregressive pretraining method. XLnet is an AR model, hence it uses the context word to predict the next word, either forward or backward[2]. However, unlike BERT [1] it only can use forward context or backward context at any time. The author aims to emphasize is that the disadvantages brought by the MASK method in BERT can be avoided using AR language model learn from bi-directional context.

To understand the gap between transformers and XLnet. We look at Transformer-XL[3]. Transformers showed incredible performance but they had limited context when compared to RNNs. As they have no notion of memory, they take fixed-length sequences as input. It had no states (that means computation can be parallelized), therefore there is an upper limit on the distance of relationships a vanilla Transformer can model. The Transformer XL is an extension of the Transformer with added recurrences at the segment level. In other words, adding a state between consecutive sequences of computations by caching the hidden states of the previous sequence and passing them as keys/values when processing the current sequence. Unlike vanilla transformers, Transformer XL uses the relative positional embedding which enables the model to learn how to compute the attention score for *n* before and after the current word.

Issues with BERT

The author of XLnet [2] points out two major problems with BERT

1. The [MASK] token used in training does not appear during fine-tuning
2. BERT predicts masked tokens in parallel. Hence it generates predictions independently.

Permutation Language Modeling

Language model consists of two phases, the pre-train phase, and fine-tune phase.[1], [2] XLNet focus on pre-train phase by proposing a new objective called Permutation Language Modeling to avoid the disadvantages brought by the MASK method in BERT. The permutation method is to get permutations of a sequence and using the previous t-1 tokens as the context to predict the t-th position token. It is trained to predict one token given preceding context like traditional language model, but instead of predicting the tokens in sequential order, it predicts tokens in some random order. The permutation can make AR model see the context from two directions.

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim Z_T}\left[\sum_{t=1}^{T} \log p_\theta(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}})\right].$$

Figure 2 Permutation language model objective

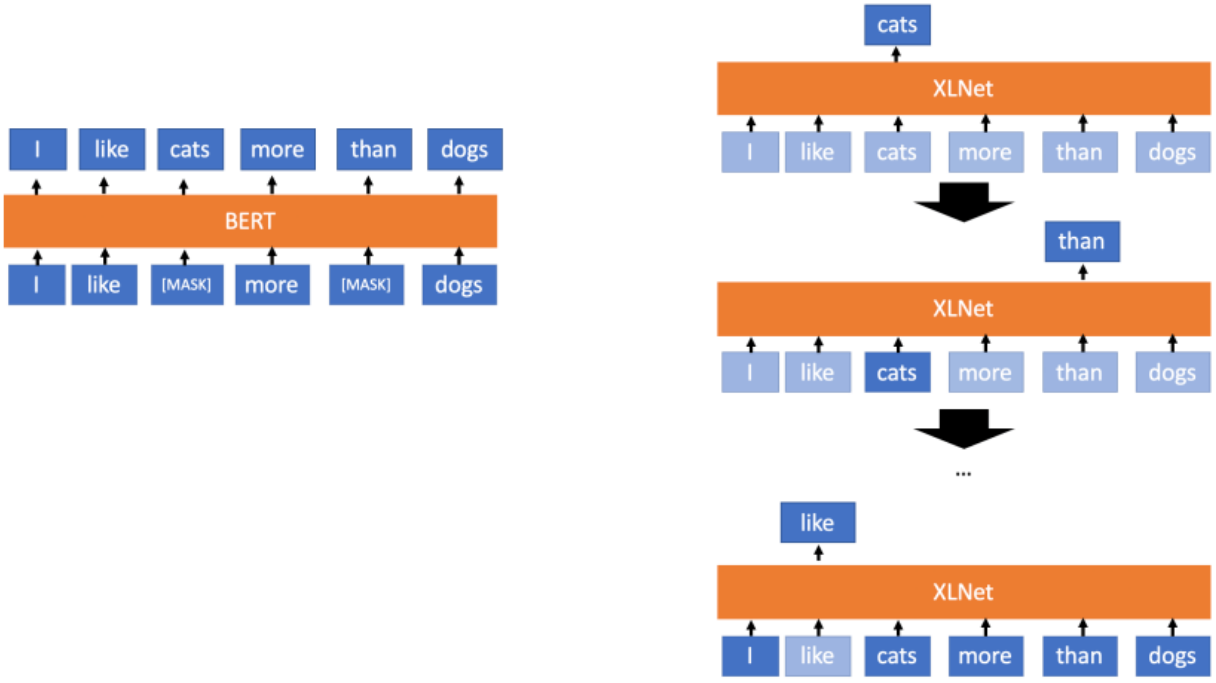


Figure 3: The conceptual difference between BERT and XLNet.

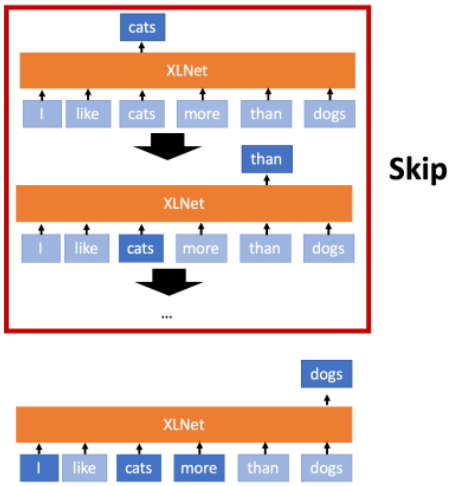
Aside from using permutation language modeling, XLNet improves upon BERT by using the Transformer XL as its base architecture. XLNet uses the two key ideas from Transformer XL: relative positional embeddings and the recurrence mechanism. The hidden states from the previous segment are cached and frozen while conducting the permutation language modeling for the current segment. Since all the words from the previous segment are used as input, there is no need to know the permutation order of the previous segment. The authors found that using the Transformer XL improved performance over BERT, even in the absence of permutation language modeling. This shows that better language models can lead to better representations, and thus better performance across a multitude of tasks, motivating the necessity of research into language modeling.

Two-Stream Self-Attention

XLNet contains two kinds of self-attention. One is the content stream attention, which is the standard self-attention in Transformer. Another is the query stream attention. Set of representations that includes both the positional embedding and token embedding is called content stream. This set of representations is used to incorporate all the information relevant to a certain word during pretraining. The content stream is used as input to the query stream, but not the other way around. XLNet introduces it to replace the [MASK] token in BERT.

Optimization

Permutation language modeling causes the model to converge slowly. To address this problem, the authors chose to predict the last n tokens in the permutation instead of predicting the entire sentence from scratch.



To enable the model to distinguish between words in different segments, BERT learns a segment embedding. In contrast, XLNet learns an embedding that represents whether two words are from the same segment. This embedding is used during attention computation between any two words.

Observations

XLNet seems to do better as it was targeted to improve BERT. I am however, interested in the tasks that would require simultaneous left to right and right to left pass. This certainly opens up a new window of opportunity for research and further developments as it has shown that it can be further improved. Although BERT and XLNet are representative models on various NLP benchmarks, they take a long time and require considerable operations to perform both training and inference. For example, training of BERT-large was performed on 64 TPUv3 chips for 4 days[1].

Comparison Study of XLNet and BERT with Large Models

XLnet team tested out BERT and XLNet on the same tasks with the same hyper parameters which were published by BERT. The following results were found [4]

Dataset	XLNet-Large (as in paper)	XLNet-Large -wikibooks	BERT-Large -wikibooks best of 3 variants
SQuAD1.1 EM	89.0	88.2	86.7 (II)
SQuAD1.1 F1	94.5	94.0	92.8 (II)
SQuAD2.0 EM	86.1	85.1	82.8 (II)
SQuAD2.0 F1	88.8	87.8	85.5 (II)
RACE	81.8	77.4	75.1 (II)
MNLI	89.8	88.4	87.3 (II)
QNLI	93.9	93.9	93.0 (II)
QQP	91.8	91.8	91.4 (II)
RTE	83.8	81.2	74.0 (III)
SST-2	95.6	94.4	94.0 (II)
MRPC	89.2	90.0	88.7 (III)
CoLA	63.6	65.2	63.7 (II)
STS-B	91.8	91.1	90.2 (III)

Interesting observations from the table:

- Trained on the same data with an almost identical training recipe, XLNet outperforms BERT by a sizable margin on all the datasets.
- The gains of training on 10x more data (comparing XLNet-Large-wikibooks and XLNet-Large) are smaller than the gains of switching from BERT to XLNet on 8 out of 11 benchmarks.
- On some of the benchmarks such as CoLA and MRPC, the model trained on more data underperforms the model trained on less data.

Predicting movie reviews with BERT on Tensorflow hub

We tried predicting the movie review sentiment using BERT. The hyperparameters were set at default. The dataset used is IMDB dataset. The data set was preprocessed in the following way.

```
Index(['sentence', 'sentiment', 'polarity'], dtype='object')
```

MAX_SEQ_LENGTH was set to 128.

Then the train and test features were converted to inputFeatures that BERT understands. For t

```
train_features = bert.run_classifier.convert_examples_to_features(train_InputExamples, label_list, MAX_SEQ_LENGTH, tokenizer)
test_features = bert.run_classifier.convert_examples_to_features(test_InputExamples, label_list, MAX_SEQ_LENGTH, tokenizer)
```

For the output layer, pooled output was used since we are classifying the entire sentence.

A custom layer was then added to tune for politeness.

```
output_weights = tf.get_variable(
    "output_weights", [num_labels, hidden_size],
    initializer=tf.truncated_normal_initializer(stddev=0.02))
```

The following parameters were set

```
BATCH_SIZE = 32
LEARNING_RATE = 2e-5
NUM_TRAIN_EPOCHS = 3.0
WARMUP_PROPORTION = 0.1
# Model configs
SAVE_CHECKPOINTS_STEPS = 500
SAVE_SUMMARY_STEPS = 100
```

Since it is a huge dataset we decided to work with 5000 training and 5000 testing samples. First, we started with 50 data samples, which as expected, did not train the network well. Then we trained the network with 500 samples. Which surprisingly yielded interesting results. It was able to classify all the sentences correctly with. Using google colab on 5000 samples. The training time was as follows.

```
Training took time  4:55:14.181930
```

Following are the evaluation results.

```
{'auc': 0.8746065,
 'eval_accuracy': 0.8746,
 'f1_score': 0.8752238,
 'false_negatives': 318.0,
 'false_positives': 309.0,
 'global_step': 468,
 'loss': 0.49648955,
 'precision': 0.8767943,
 'recall': 0.87365913,
 'true_negatives': 2174.0,
 'true_positives': 2199.0}
```

Test accuracy being 0.8746 which is about 87.46%

For sentence predictions we tried predicting the following sentences.

```
pred_sentences = [
  "I found the movie to be very bad and disappointing",
  "The movie was ok",
  " It was a beautiful sunny day",
  " it was very confusing and lacking in plot",
  " The cast did a good job!",
  "I wasted my money on this",
  "The previous movie was better than this",
  "My expectations were not met "
]
```

The results it yielded are as follows

```
[('I found the movie to be very bad and disappointing',
  array([-8.9914893e-04, -7.0145388e+00], dtype=float32),
  'Negative'),
 ('The movie was ok',
  array([-0.00661422, -5.021832  ], dtype=float32),
  'Negative'),
 (' It was a beautiful sunny day',
  array([-4.152661  , -0.01584745], dtype=float32),
  'Positive'),
 (' it was very confusing and lacking in plot',
  array([-8.255411e-04, -7.099944e+00], dtype=float32),
  'Negative'),
 (' The cast did a good job!',
  array([-5.3407264e+00, -4.8039020e-03], dtype=float32),
  'Positive'),
 ('I wasted my money on this',
  array([-7.5752643e-04, -7.1858678e+00], dtype=float32),
  'Negative'),
 ('The previous movie was better than this',
  array([-0.5590346 , -0.84807336], dtype=float32),
  'Negative'),
 ('My expectations were not met ',
  array([-0.0172826, -4.066683 ], dtype=float32),
  'Negative')]
```

Predicting movie reviews with XLNet

Our attempt at running at 5000 samples

```
INFO:tensorflow:global_step/sec: 1.77529
I0504 18:12:45.257639 140593394272128 basic_session_run_hooks.py:692] global_step/sec: 1.77529
INFO:tensorflow:loss = 0.0884195, step = 3900 (56.328 sec)
I0504 18:12:45.258563 140593394272128 basic_session_run_hooks.py:260] loss = 0.0884195, step = 3900 (56.328 sec)
INFO:tensorflow:Saving checkpoints for 4000 into exp/imdb/model.ckpt.
I0504 18:13:41.022281 140593394272128 basic_session_run_hooks.py:606] Saving checkpoints for 4000 into exp/imdb/model.ckpt.
2020-05-04 18:13:53.672428: W tensorflow/core/framework/op_kernel.cc:1651] OP_REQUIRES failed at save_restore_v2_ops.cc:134 : Resource exhausted: exp/imdb/model
Traceback (most recent call last):
  File "/tensorflow-1.15.2/python3.6/tensorflow_core/python/client/session.py", line 1365, in _do_call
    return fn(*args)
```

For training with XLNet we used the same IMDB dataset with the same hyperparameter configuration.

The tokenizing of sentences are as follows.

```
Tokenize the first sentence:
['_vampires', ',', '_sexy', '_guys', ',', '_guns', '_and', '_some', '_blood', '.', '_who', '_could', '_ask', '_for',
```

We then create the Attention masks. For decay we disable it.

```
MAX_LEN = 128
input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post")
```

```
# Create attention masks
attention_masks = []

# Create a mask of 1s for each token followed by 0s for padding
for seq in input_ids:
  seq_mask = [float(i>0) for i in seq]
  attention_masks.append(seq_mask)
```

```
param_optimizer = list(model.named_parameters())
no_decay = ['bias', 'gamma', 'beta']
optimizer_grouped_parameters = [
    {'params': [p for n, p in param_optimizer if not any(nd in n for nd in no_decay)],
     'weight_decay_rate': 0.01},
    {'params': [p for n, p in param_optimizer if any(nd in n for nd in no_decay)],
     'weight_decay_rate': 0.0}
]
```

Since our attempt at running the program at 5000 failed terribly. We thought we'll start from 50. The following results are obtained on training the dataset on 50 samples.

```
Epoch:   0%|          | 0/4 [00:00<?, ?it/s]/usr/local/lib/python3.6/dist-packages/torch/nn/functional
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh instead.")
/pytorch/torch/csrc/utils/python_arg_parser.cpp:756: UserWarning: This overload of add_ is deprecated:
        add_(Number alpha, Tensor other)
Consider using one of the following signatures instead:
        add_(Tensor other, *, Number alpha)
Epoch:  25%|██▌       | 1/4 [00:55<02:45, 55.23s/it]Train loss: 0.9104541838169098
Epoch:  50%|█████     | 2/4 [01:47<01:48, 54.47s/it]Train loss: 0.6784721612930298
Epoch:  75%|███████▌  | 3/4 [02:39<00:53, 53.61s/it]Train loss: 0.5208148509263992
Epoch: 100%|██████████| 4/4 [03:31<00:00, 52.76s/it]Train loss: 0.5544064044952393
```

```
Test Accuracy of the finetuned model on val data is: 50.0 %
```

Now, we try to train the model with 1000 samples

```
Epoch:  25%|██▌       | 1/4 [17:15<51:46, 1035.63s/it]Train loss: 0.6971887898445129
Epoch:  50%|█████     | 2/4 [34:30<34:30, 1035.36s/it]Train loss: 0.4920054489374161
Epoch:  75%|███████▌  | 3/4 [51:50<17:16, 1036.77s/it]Train loss: 0.2951955908536911
Epoch: 100%|██████████| 4/4 [1:09:11<00:00, 1037.90s/it]Train loss: 0.18319756403565407
```

The test accuracy came out to be

```
Test Accuracy of the finetuned model on val data is: 80.0 %
```

It definitely doesn't do as well as BERT here. But it may be due to the sample size. Nothing definitive can be said about it unless we run it on the same sample size.

Comparison of BERT and XLNet on Computational Basis.

The paper 'Comparing BERT and XLNet from the Perspective of Computational Characteristics'[5] provides a thorough comparison. Since we lack the resources to compute this, we decided to use the previous results. The paper suggests that the both models exhibit similar computational characteristics except the target-position-aware representation and relative position encoding features of XLNet, leading to a better benchmark score at the cost of 1.2× arithmetic operations and 1.5× execution time on a modern CP. The author used PyTorch 1.2.0 as a reference framework, Intel Xeon Gold 6138 processor (Skylake SP), and MRPC benchmark of GLUE (General Language Understanding Evaluation[6]) datasets. Most popular operations are Mat-Mat mul, which takes up most of the execution time of the both models. Mat-Mat mul, which depends on the characteristics of the data used in the calculation, can be divided into Mat-Mat mul (1) multiplying an input feature by weight, and Mat-Mat mul (2) multiplying an input feature by another input feature. Mat-Mat mul (1) is a typical compute-intensive operation that fully utilizes the cache, so Mat-Mat mul (1) has a large FLOP/B. Mat-Mat mul (2) has a smaller input feature than Mat-Mat mul (1), and the input feature is used only once in Mat-Mat mul (2). It leads to higher memory bandwidth requirement and low FLOP/B compared to Mat-Mat mul (1). Also, XLNet exhibits a better benchmark score at the cost of 1.2× through additional target position-aware representation and relative position encoding features.

Conclusion and Future works

We are not nearly close to being done. Since these pretraining models are new, there is a lot of scope for improvement and experimentation. Perusing the comparative studies already done, we were only able to find one paper other than Google's own fair comparison study [4]. There is a gap that can be worked with.

Furthermore, different analysis can be done, like aspect based sentiment analysis [7], on XLNet to compare the results with the ones done using BERT.

It is evident that XLNet outperforms BERT. But is it always the case? This is the question we are interested in exploring.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *ArXiv181004805 Cs*, May 2019, Accessed: May 04, 2020. [Online]. Available: http://arxiv.org/abs/1810.04805.

[2] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, “XLNet: Generalized Autoregressive Pretraining for Language Understanding,” *ArXiv190608237 Cs*, Jan. 2020, Accessed: May 04, 2020. [Online]. Available: http://arxiv.org/abs/1906.08237.

[3] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context,” *ArXiv190102860 Cs Stat*, Jun. 2019, Accessed: May 04, 2020. [Online]. Available: http://arxiv.org/abs/1901.02860.

[4] Xln. Team, “A Fair Comparison Study of XLNet and BERT with Large Models,” *Medium*, Jul. 22, 2019. https://medium.com/@xlnet.team/a-fair-comparison-study-of-xlnet-and-bert-with-large-models-5a4257f59dc0 (accessed May 04, 2020).

[5] H. Li, J. Choi, S. Lee, and J. H. Ahn, “Comparing BERT and XLNet from the Perspective of Computational Characteristics,” in *2020 International Conference on Electronics, Information, and Communication (ICEIC)*, Jan. 2020, pp. 1–4, doi: 10.1109/ICEIC49074.2020.9051081.

[6] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding,” *ArXiv180407461 Cs*, Feb. 2019, Accessed: May 04, 2020. [Online]. Available: http://arxiv.org/abs/1804.07461.

[7] C. Sun, L. Huang, and X. Qiu, “Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence,” 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)*, Minneapolis, Minnesota, Jun. 2019, pp. 380–385, doi: 10.18653/v1/N19-1035.