How does a Generalized Autoregressive Pretraining Method Compare to an Autoencoder Language Model on Natural Language Processing Tasks?

Khushaal Singh Jammu

Subject: Computer Science

Word Count: 3889

Contents

Section 1

Introduction

The research question of this paper is: **"How does a Generalized Autoregressive Pretraining Method Compare to an Autoencoder Language Model on Natural Language Processing Tasks?"** The potential scope of the paper is enormous due to the broad nature of the field of natural language processing. Thus, to make the problem more tractable, two natural language tasks will be selected to evaluate on: reading comprehension and semantic similarity.

1.1 Dissecting the Research Question

Much of the vocabulary involved in this paper is out of the scope of the syllabus. It is therefore useful to define them before proceeding.

This paper falls under the field of artificial intelligence, a subset of computer science which attempts to simulate human-level intelligence using computer systems, specifically deep learning (Russell and Norvig 2009).[1]

1. Please refer to Appendix A for an important introduction to deep learning theory.

The specific subfield of artificial intelligence in which this paper engages is natural language processing (NLP)[2], which is concerned with the interaction between computers and human language, specifically the means by which computers could be endowed with understanding of human language (Garbade 2018).

Due to the broad nature of this field, this paper selects two different representative tasks: reading comprehension and semantic similarity. Reading comprehension is a question-answering task in which a computer system attempts to answer a question based on a short passage of text; it is alternatively referred to as machine comprehension (Rajpurkar et al. 2016). Semantic similarity is a task wherein a computer system must determine how similar two given pieces of text are based on their meaning (Harispe et al. 2015).

This paper is concerned with different approaches to natural language processing, and, by extension, the tasks enumerated above. This paper compares two different approaches: the generalized autoregressive pretraining method versus the autoencoder language model approach.

1.2 The Significance of Tasks Selected

The ability for machines to understand human language extends far beyond a technical challenge. A system capable of understanding and communicating in human language would be revolutionary. It would have a wide scope of application in a diverse set of fields, ranging from an automated research assistant that helps a lawyer prepare for a case to an education expert who

2. Please refer to Appendix B.1 for an introduction to NLP.

writes questions for an exam (Monsters 2017).

This paper selected the two tasks that it did because they are broadly representative of some of the major ways in which humans comprehend language. Reading comprehension was selected because answering questions on a body of text requires thorough semantic understanding much like a human would (Rajpurkar et al. 2016). Semantic similarity is similar in that regard, requiring a structural understanding of the text and its content to perform well. However, it differs in that it: (a) tests the understanding of short snippets of text, and (b) requires an appreciation of the nuances and subtleties in the language used (Harispe et al. 2015).

This paper evaluates two different, yet fundamentally similar, approaches to NLP tasks. The potential for a machine that is capable of understanding and using language at the same level as humans is a driving force for exploring the research question.

1.3 Outline of This Paper

Deep learning, natural language, and reading comprehension encompass a vast range of candidates for the research question. In this paper, for the purposes of making the investigation more feasible, only a few representative algorithms/tasks are selected.

The two approaches to NLP tasks selected are: Bidirectional Encoder Representations from Transformers, abbreviated BERT, and XLnet. Refer to the "Justifiation of Algorithms" for why these were selected.

These will be tested on reading comprehension on the Stanford Question Answering Dataset 2.0 and semantic similarity on the Semantic Textual

Similarity Benchmark from the General Language Understanding Evaluation collection of benchmarks. Specifically, both BERT and XLnet will be fine-tuned separately for each of these tasks on three different platforms: one with extremely high resource availability, one with medium, and one with comparatively little. This will be done to measure how well each model performs in resource-constrained environments, a key factor when deciding whether or not to work with a deep learning model.

Statistics regarding each approach will be collected and a thorough analysis of both will follow such that a conclusion can be made on the comparative suitability of one versus another (e.g. in terms of accuracy, resource consumption, time taken to train).

It is hypothesized that XLnet will outperform BERT in terms of absolute accuracy, but that this will come at the cost of significantly greater resource demands during pre-training.

Section 2

Justification of Algorithms Used

This paper compares the effectiveness of two different approaches to NLP tasks: BERT and XLnet. This chapter will expand on the thought-process behind their selection. Refer to Appendix B.3 and B.4 for a formal description of how they work.

2.1 Justification for Tasks Selected

In the field of NLP, there are a plethora of different tasks and sub-fields. In order to make this investigation more practical, this paper decided to limit the scope of our investigation to just the tasks of reading comprehension and semantic similarity.

Reading comprehension is widely used among humans to test for semantic understanding in standardized testing such as the SAT or ACT. It follows therefore, that it should be useful for doing the same for computer systems. A

more theoretical justification can be found in the fact that in order to perform reading comprehension of a difficult or convoluted passage, the model must first be able to understand it (Rajpurkar et al. 2016). Thus, it provides a good measure for how capable a model is at understanding language.

This paper selected semantic similarity for a similar reason; namely that being able to tell how similar two texts are requires semantic understanding of the text's content itself, albeit with far greater granularity (Harispe et al. 2015).

2.2 Justification for Datasets Selected

The dataset used for reading comprehension is the *Stanford Question Answering Dataset 2.0* (Rajpurkar, Jia, and Liang 2018), henceforth referred to as **SQuAD**. This paper selected it because it is currently the standard dataset for measuring a computer system's capability to answer questions based on a passage of text (Rajpurkar, Jia, and Liang 2018). It consists of questions on a set of Wikipedia articles where the answer might be a segment of text from the corresponding passage, or otherwise unanswerable. The inclusion of unanswerable questions is its primary improvement over its predecessor SQuAD 1.1. The inclusion of unanswerable questions is important because given a passage of text, there is a finite set of questions which can be answered from it, and it may be that said passage doesn't contain the requisite information to answer a question. Humans are capable of diserning when that happens, and a computer system that attempts reading comprehension should not begin with the faulty assumption that all questions asked about a text must be capable of being answered (Rajpurkar, Jia, and Liang

2018).

The dateset used for semantic similarity is the *Semantic Textual Similarity Benchmark*, henceforth referred to as the **STS** benchmark, from the *General Language Understanding Evaluation*, henceforth referred to as **GLUE**, collection of benchmarks (Conneau and Kiela 2018)). GLUE is a collection of benchmarks for various NLP tasks, ranging from question-answering to summarization. Its purpose is to drive research for the development of general and robust NLP systems. The STS benchmark is a selection of English datasets used in semantic similarity tasks organized as part of Semantic Evaluation (henceforth SemEval), which is an ongoing series of evaluations of computational semantic analysis systems (*SemEval-2020*), from 2012 and 2017 . This paper selected it because (1) it is the industry standard benchmark for evaluating a model's ability to compute semantic similarity (*SemEval-2020*), (2) it contains a large amount of data, and (3) it includes a diverse selection of datasets including from image captions, news headlines, and online forums.

2.3 BERT

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (Devlin et al. 2018). Developed by researchers at Google, it is designed to train deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers. Consequently, the pre-trained BERT model is capable of being fine-tuned with just a single additional output layer to create state-of-the-art (SOTA) models for a wide range of tasks, such as reading comprehension, as is relevant to this paper.

At the time of its introduction, it obtained new SOTA results on 11 natural language processing tasks (Devlin et al. 2018).

Its key technical innovation is applying the bidirectional training of a transformer architecture, an attention model, to language representation (Devlin et al. 2018). This contrasts with previous conditional left-to-right, right-to-left, or some mixture of the two. It was selected because their paper shows that such a model can have a deeper understanding of language context and flow compared to a single-direction language model. Other candidates such as Facebook's Document Retrival Question Answering system were considered too; however, BERT was selected as it was, and still is, a conceptually and empirically groundbreaking innovation and large amounts of subsequent work has been built on it; the original paper has more than eight and a half thousand citations (Devlin et al. 2018).

2.4 XLnet

XLnet is a generalized autoregressive pretraining method that both enabes learning bidirectional contexts and overcomes the perceived limitations of BERT thanks to its autoregressive formulation (Yang et al. 2019). It further integrates ideas from Transformer-XL, the SOTA autoregressive model, into pretraining. Note that Transformer-XL is an improved version of the Transformer architecture which BERT is based on.

Its key technical innovation is to realise that BERT suffers from a pretrain-finetune disrepancy (Yang et al. 2019). This is because during BERT's pretraining, it masks random tokens and attempts to predict what they are. However, no such equivalent exists during finetuning. A further consequence

of corrupting the input with masks is that BERT neglects dependency between masked positions. This paper selected XLnet because it attempts to rectify BERT's shortcomings. In doing so, it outperforms BERT on 20 different language modelling tasks (Yang et al. 2019), but this is shown to come at a significant resource cost. It is interesting, therefore, to investigate whether the increased performance is worth its drawbacks.

Section 3

Method

3.1 Procedure

The experiments conducted in this paper involve fine-tuning the publicly available pre-trained models of BERT and XLnet, fetching certain statistics for analysis. In order to collect resource consumption statistics, open source tools such as *nvidia-smi* and Tensorboard were used.

At each experiment, both XLnet and BERT were finetuned for each of the three language modelling tasks. In order to compare their suitability in resource-constrained environments, each experiment was conducted once in three different environments[1]: a Google Cloud (GCloud) Instance with a v3-8 TPU, a GCloud Instance with a 16GB GPU, and a GCloud Instance with an 8GB GPU. Therefore, there were 12 different trials that give insight into the performance of each algorithm with varying resource availability.

After setup, the general program flow of each experiment is as follows:

- **Preprocess dataset** locally so it's ready for training

1. These are the Powerful, Medium, and Small platforms respectively.

- **Load preprocessed training dataset** from either the local or remote filesystem for usage during training
- **Perform fine-tuning** for a given using existing implementations
- **Measure statistics** during the training process using tools such as Tensorboard
- **Output results and analysis**

3.2 Datasets

3.2.1 SQuAD

This paper uses SQuAD as the designated dataset for the task of machine comprehension. Released and maintained by Stanford University, it consists of questions on a set of Wikipedia articles.

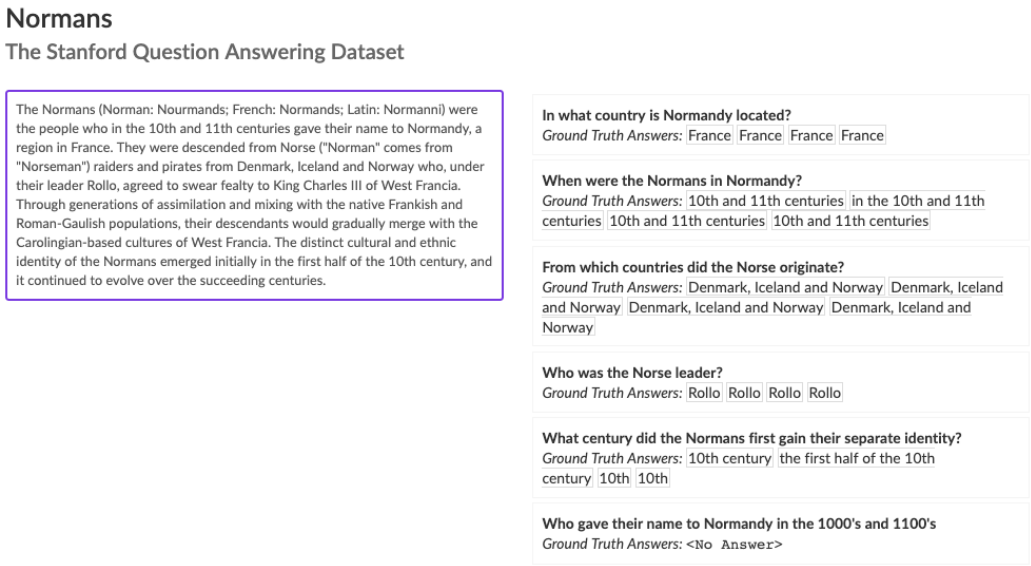


Figure 3.1: Example of a SQuAD passage and associated questions

Figure 3.1 depicts an example of what a SQuAD passage and the associated question-answer pairs might look like. Observe how for the first paragraph in the text entitled "Normans," there are six different questions, one of which is unanswerable. In practise, our models are fed passage-question pairs and are trained to output an answer where one can be found. There are a total of 130,319 questions across 19,035 passages in the training set.

3.3 Environments

All experiments were conducted on a prebuilt deep learning image provided by Google. All the models were implemented in one of two software environments: Tensorflow 1.4 and Python 2, or PyTorch and Python 3. TensorFlow is an open source library for numerical computation and large-scale machine learning. It was the tool of choice for this paper because the publicly released implementations of both BERT and XLNet were written in Tensorflow. PyTorch is another popular open source library and it was selected because there are publicly released implementations of both the models that work in resource-constrained environments.

3.4 Implementations

Companies like Google have open-sourced implementations to their deep learning models, usually written in Tensorflow or PyTorch. In industry, it makes little sense to reimplement them in a custom library.

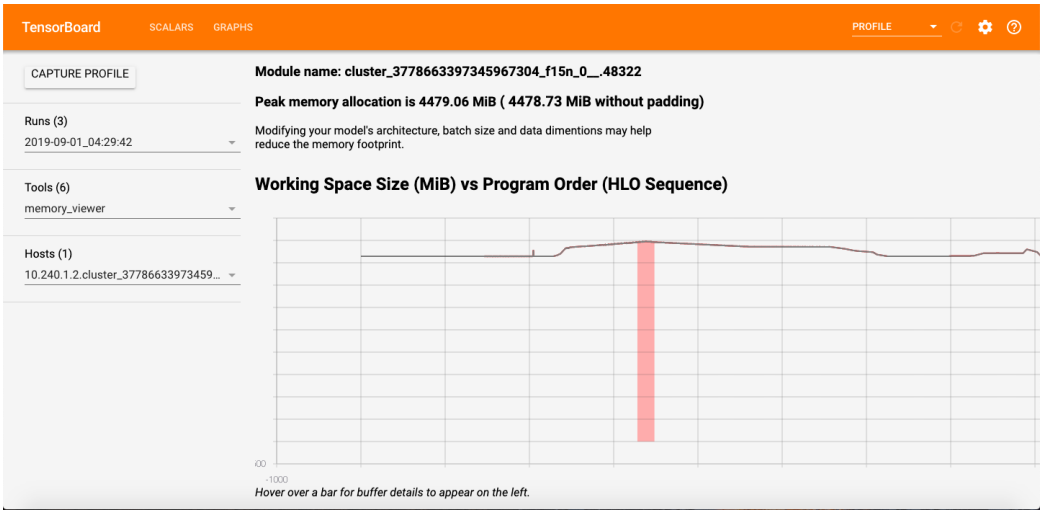
Accordingly, this paper uses the publicly available implementations and pretrained models of XLNet and BERT, as released by their respective au-

thors. However, the code was modified wherever appropriate to include a profiler.

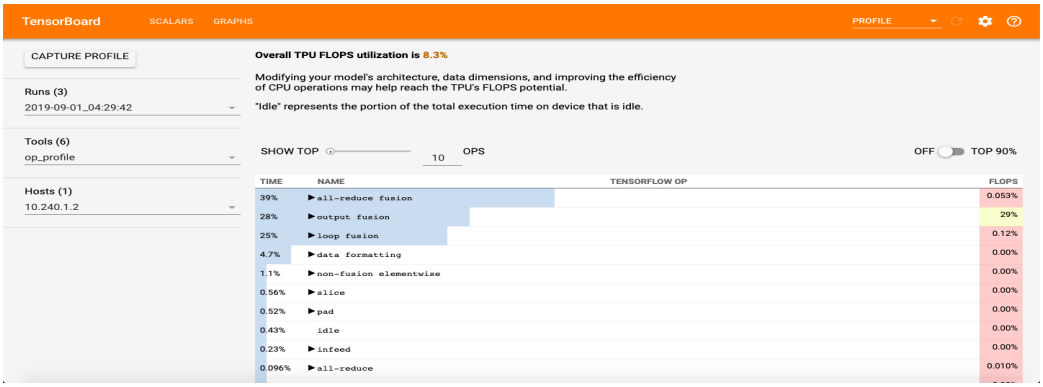
3.5 Measurements

The following measurements were made each trial and then recorded:

- **Overall TPU/GPU Utilization** to measure the intensiveness of training process
- **Peak Memory Allocation** to quantify the memory footprint
- **Training Duration** to measure the time taken for training to elapse
- **Final Evaluation Score** to measure the final performance of the model



(a) Memory Allocation



(b) TPU Utilization

Figure 3.2: Tensorboard Profiling Visualiser

3.6 Important Considerations

With regard to XLNet, this paper made use of the *XLNet-Large, Cased* pretrained model. With regard to BERT, this paper made use of the *BERT-Large, Cased (Whole Word Masking)* pretrained model. Both models share the same architecture hyperparameters, with 24 layers, a hidden size of 1024, and 16 attention heads. However, owing to their different underlying structure, their total number of hyperparameters differs. In both cases, the default hyperparameters suggested by the authors of their respective papers were used. Additionally, all data used as preprocessed was suggested where appropriate.

Each experiment was conducted in isolation and in the absence of any other running programs to control as much as possible, for otherwise it would be impossible to determine the program's true efficiency. Furthermore, this paper followed the Tensorflow official recommendation of profiling a minimum number of times to avoid slowing down the program (*TensorFlow Core*), wherever relevant.

For experiments conducted on the medium and small instances, PyTorch implementations of BERT and XLnet were employed, because they are more suitable for training in resource-constrained environments. The PyTorch implementations were able to obtain SOTA results for all the tasks concerned (Huggingface 2019); thus, this paper assumes that there is no performance difference between it and the original Tensorflow implementations.

During evaluation, all of the model's checkpoints were evaluated as opposed to just the final checkpoint, in order to avoid the problem of overfitting the training dataset.

Section 4

Results and Analysis

Platform	**Model**	**Task**	**TPU/GPU Utilization**	**Peak Memory (MiB)**	**Duration (min)**	**Evaluation Score**
Powerful	XLnet	SQuAD	21.00%	13600	69	88.6 (F1)
Powerful	BERT	SQuAD	28.00%	66700	31	83.2 (F1)
Powerful	XLnet	STS-B	8.30%	4480	8	0.911 (Pearson R)
Powerful	BERT	STS-B	20.00%	4510	7	0.841 (Pearson R)
Medium	XLnet	SQuAD	97.00%	14900	122	63.1 (F1)
Medium	BERT	SQuAD	96.00%	7970	505	79.0 (F1)
Medium	XLnet	STS-B	96.00%	5590	62	0.706 (Pearson R)
Medium	BERT	STS-B	96.00%	5400	55	0.820 (Pearson R)
Small	XLnet	SQuAD	95.00%	15500	5230	52.0 (F1)
Small	BERT	SQuAD	98.00%	7990	2170	78.2 (F1)
Small	XLnet	STS-B	97.00%	6000	1030	0.654 (Pearson R)
Small	BERT	STS-B	99.00%	5862	966	0.792 (Pearson R)

Table 4.1: Raw data for overall TPU/GPU utilization, peak memory allocation, training duration, and evaluation score for each of the respective platforms and tasks. The evaluation scores were calculated using the appropriate methodology for each task after the training completed.

Table 4.1 shows the raw data for both reading comprehension on SQuAD and semantic similarity evaluation on STS-B. These results were collected during the training process, except for the evaluation score which was calcu-

lated after training completed.

The analysis will be broken down into each category of measurement.

4.1 Overall TPU/GPU Utilization

As shown in Table 4.1, the overall TPU utilization during training varies significantly for the experiments conducted on the powerful platform, with an absolute range of 19.7% and an average of 19.3%. By contrast, the overall GPU utilization for experiments conducted on both the medium and small platforms was roughly the same, averaging 96.8% with an absolute range of 4%.

The most likely explanation for the this stark difference could be that the TPU's computational capabilities far exceeds what is demanded of it in the fine-tuning process. The authors of the XLnet and BERT implementations might not have bothered optimising the fine-tuning process as much as could have been done because its speed renders such overoptimization redundant, with the time being better spent optimizing the pre-training code which does take a significant amount of time.

Alternatively, it might be that the PyTorch implementations of the two models is more efficient than the Tensorflow one. However, this is unlikely. Firstly, Tensorflow was developed by Google Brain, an organization involved in the creation of both XLnet and BERT. It is much more likely that they would design their model to take advantages of the optimizations that Tensorflow offers, which might not be easily reproduced in PyTorch. Secondly, the PyTorch implementations of both XLnet and BERT were able to achieve SOTA performance on both SQuAD and GLUE (Huggingface 2019). There

cannot, therefore, be any issues with regard to accuracy, and given that they are implementation libraries, it's unlikely that the performance would be dissimilar.

4.2 Peak Memory Allocation

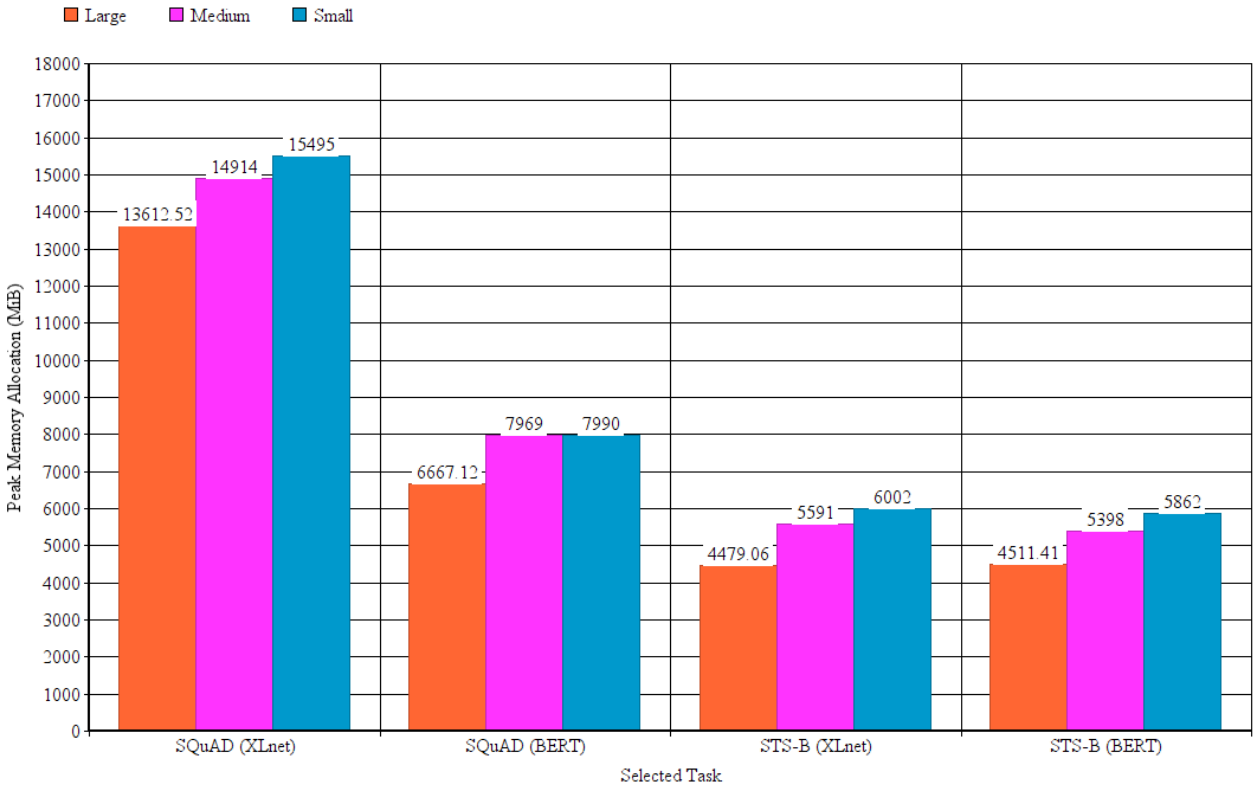


Figure 4.1: Graph of peak memory allocation for SQuAD and STS-B on each platform for both models.

As shown in the graph in Figure 4.1, the peak memory allocation is significantly higher, almost double the other values, for SQuAD (XLnet). The SQuAD tasks require more than the STS-B tasks, and there appears to be a general trend that the more powerful the platform, the higher the

peak memory allocation. It must also be noted, however, that the memory allocation was constant throughout the training process. This is because only the training data is loaded into the primary memory, which would take up a fixed amount of space. Therefore, the variations in peak memory allocation are likely to do with the padding and other hyperparameters.

4.3 Training Duration

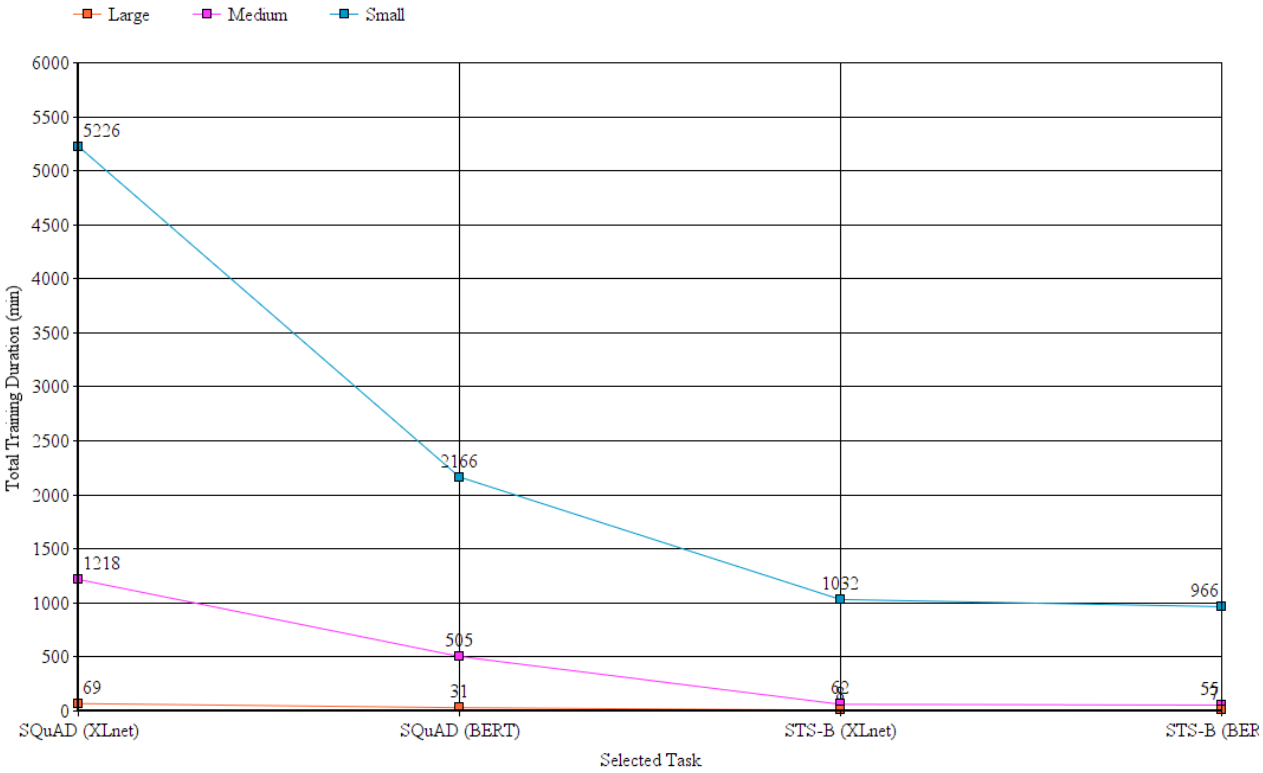


Figure 4.2: Graph of total training duration for SQuAD and STS-B on each platform for both models.

As shown in the graph in Figure 4.2, training BERT for reading comprehension on the SQuAD dataset takes significantly less time that training

XLnet on it. Training BERT for STS-B does take less time than XLnet, however the difference is minimal.

The most striking characteristic of the graph is not the differences between tasks, but the differences between various platforms. For example, while fine-tuning XLnet for SQuAD only takes 69 minutes on the large platform, it takes 5226 minutes on the small platform. That is a difference of almost 86 hours! This trend is clearly visible across all four tasks, with training on the small platforms taking significantly longer than on the medium & large platforms. This is because as the power of the TPU/GPU varies, the training hyperparameters must be adjusted, because if the memory capacity of the training device differs, then the size of training data that can be processed at once will differ as well. A TPU can hold significantly more training data in a single batch than most GPUs, owing to its 32GB of high-bandwidth memory. If less training data can be processed per iteration, then more iterations will be required to reach a similar level of performance. Therefore, training on less powerful platforms, e.g. the small platform, will take significantly longer than on more powerful platforms, e.g. the large one. This suggests that: (1) it is faster to train on more powerful machines and (2) BERT is generally faster to train than XLnet.

4.4 Evaluation Score

The graph in Figure 4.3 shows the evaluation scores for each of the experiments. As expected, XLnet outperforms BERT on each of the large platform for all four tasks. However, while the BERT scores decrease slowly across less powerful platforms, the XLnet scores drop sharply. XLnet's SQuAD perfor-

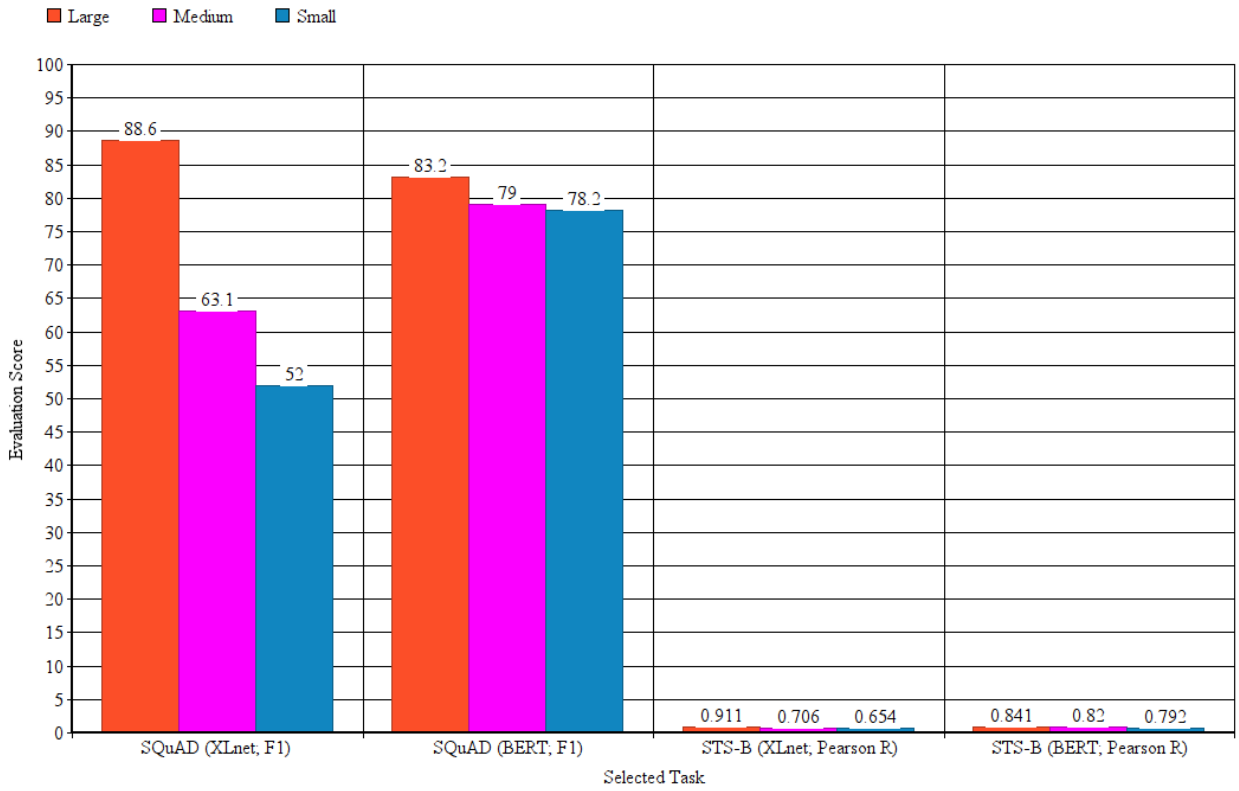


Figure 4.3: Graph of evaluation for SQuAD, which uses the F1 metric, and STS-B, which uses the Pearson R metric, on each platform for both models. Note that the scale might be misleading, because the range of an F1 score is $0 \leq x \leq 100$ [1], whereas the range for a Pearson R coefficient is $-1 \leq x \leq 1$.

mance on the small platform is only 66.5% of the corresponding BERT score. This suggests that although XLnet performs very well with large amounts of computing capability, its performance rapidly degrades as the resources available decrease.

1. The F1 scores has been transformed (multiplied by 100) to conform with the current convention. The range would otherwise be $0 \leq x \leq 1$.

Section 5

Conclusion

To restate, the research question this paper attempts to answer is: **"How does a Generalized Autoregressive Pretraining Method Compare to an Autoencoder Language Model on Natural Language Processing Tasks?"** The selected generalized autoregressive pretraining method is XLnet, and the selected autoencoder language model is BERT.

It can be concluded from this study's experimental results that the suitability of either of the models is dependent on the context in which it will be used. This paper's findings include that from a pure performance standpoint, XLNet consistently outperforms BERT on both the tasks, given sufficient computational resources. Furthermore, its implementation has many optimizations which take advantage of the unique features TPUs offer in order to speed up training. For use-cases where resource availability is not an issue, XLnet would clearly be the better choice.[1] However, it must be noted that training XLnet is extremely resource intensive; far more so than BERT. This

1. One possible exception would be if the project relies on external third-party libraries; there are many publicly available libraries which apply BERT to various NLP tasks which do not exist for XLNet.

is most clearly visible in its sharp drop in evaluation scores for the medium and small instances. BERT by contrast, although underperforming slightly for trials conducted on the large instance, outperforms XLnet by a significant margin for trials conducted on the medium and small instances. This suggests that it is possible to adequately train BERT in resource-constrained environments. Moreover, BERT does converge in less time than XLnet, making it a more suitable choice wherever time is a limiting factor.

Overall, this study concludes that the generalized autoregressive pretraining method obtains superior performance on language modelling tasks than the autoencoder language mode. However, the latter is preferable for usage in resource-constrained environments because its training is not unduly hampered, whereas the former's training definitely is.

5.1 Evaluation

Hyperparameter Selection

This paper made use of the suggested hyperparameters wherever possible, except when there were insufficient resources. However, these suggestions are often only applicable for a specific environment, and is unsuitable for training with less computational power (Zihangdai 2019). This could have detrimentally influenced the results obtained. In practise, further experimentation for selecting the appropriate hyperparameters is required.[2]

Additionally, all trials were conducted with two epochs of training. The models may have converged to an approximated desired level earlier, which

2. This specifically considers hyperparameters influencing the size of each iteration, for example the batch size, sequence length, and document stride.

would render the rest of the training superfluous. Alternatively, it may be that a longer training duration is required to achieve optimal performance.

Cost of Training

All experiments and development was conducted on instances rented on the Google Cloud Platform, using subsidised cloud funding. As a result, there was no monetary cost incurred. However, this will not be the case for the majority of people. Therefore, training cost is a serious consideration which this paper does not take into account.

5.2 Implications of Results

This paper does not claim that one approach to natural language processing is universally superior to the other. Rather, it must be kept in mind that the suitability of different approaches depends on the tasks at hand, as well as resource availability, cost tolerance, and time. Additionally, there may be other edge cases and other factors that this paper did not consider.

Bibliography

Alammar, Jay. *The Illustrated Transformer.* http://jalammar.github.io/illustrated-transformer/.

Brownlee, Jason. 2019. *A Gentle Introduction to Transfer Learning for Deep Learning.* https://machinelearningmastery.com/transfer-learning-for-deep-learning/.

Conneau, Alexis, and Douwe Kiela. 2018. *SentEval: An Evaluation Toolkit for Universal Sentence Representations.* arXiv: 1803.05449 [cs.CL].

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv: 1810.04805 [cs.CL].

Flach, Peter A. 2017. *Machine learning: the art and science of algorithms that make sense of data.* Cambridge University Press.

Garbade, Michael J. 2018. *A Simple Introduction to Natural Language Processing.* https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32.

Glen, Stephanie. 2017. *Dimensionality & High Dimensional Data: Definition, Examples, Curse of.* https://www.statisticshowto.datasciencecentral.com/dimensionality/.

Goldberg, Yoav. 2017. *Neural network methods for natural language processing.* Morgan & Claypool.

Goodfellow, Ian. 2016. *Deep Learning.* MIT Press.

Harispe, Sébastien, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. 2015. "Semantic Similarity from Natural Language and Ontology Analysis." *Synthesis Lectures on Human Language Technologies* 8 (1): 1–254. ISSN: 1947-4059. doi:10.2200/s00639ed1v01y201504hlt027. http://dx.doi.org/10.2200/S00639ED1V01Y201504HLT027.

Huggingface. 2019. *huggingface/transformers.* https://github.com/huggingface/transformers.

Monsters, Data. 2017. *10 Applications of Artificial Neural Networks in Natural Language Processing.* https://medium.com/@datamonsters/artificial-neural-networks-in-natural-language-processing-bcf62aa9151a.

Packt. *Feed-forward and feedback networks.* https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec21/feed-forward-and-feedback-networks.

Radhakrishnan, Pranoy. 2017. *What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?* https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a.

Rajpurkar, Pranav, Robin Jia, and Percy Liang. 2018. *Know What You Don't Know: Unanswerable Questions for SQuAD.* arXiv: 1806.03822 [cs.CL].

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. *SQuAD: 100,000+ Questions for Machine Comprehension of Text.* arXiv: 1606.05250 [cs.CL].

Russell, Stuart J., and Peter Norvig. 2009. *Artificial intelligence.* Prentice Hall.

SemEval-2020. http://alt.qcri.org/semeval2020/index.php?id=tasks.

Sewaqu. 2010. *File:Linear regression.svg.* https://commons.wikimedia.org/wiki/File:Linear_regression.svg.

TensorFlow Core. https://www.tensorflow.org/guide/.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. 2015. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.* arXiv: 1502.03044 [cs.LG].

Yang, Zhilin, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. *XLNet: Generalized Autoregressive Pretraining for Language Understanding.* arXiv: 1906.08237 [cs.CL].

Zihangdai. 2019. *zihangdai/xlnet.* https://github.com/zihangdai/xlnet.

Appendices

These appendices will endeavour to provide background information about relevant concepts and academic fields, so as to make this paper more accessible to readers without a background in Artificial Intelligence. However, they should not be taken as comprehensive, and readers are encouraged to explore that which interests them elsewhere.

Appendix A

Introduction to Deep Learning Theory

Artificial intelligence (AI) is a field of study concerned with the development of computer systems that can mimic the cognitive functions associated with the human mind and, by extension, human intelligence itself (Russell and Norvig 2009).

A.1 Machine Learning

Machine learning, alternatively abbreviated ML, is the systematic study of algorithms and systems that can acquire knowledge by examining large volumes of data and, by approximating mathematical models, infer patterns from said data to make predictions on future events (Flach 2017).

Machine learning algorithms (sometimes referred to as models, because they model the relationship between the input and output data) typically consist of:

- A task T that we want the algorithm to perform well at. Examples of T include classifying tumours into one of two categories (benign and malignant), fraud detection where anomalous transactions are identified and flagged, and transcribing a speech to generate captions from a video of a speech. There are two broad categories of machine learning tasks: classification and regression. In classification, the algorithm is asked to specify which of k categories some input belongs to. With regression, the challenge is to predict some numerical value given some input.

- Some quantitative performance measure P that measures the algorithm's capability to execute T. This usually entails the use of a mathematical *cost function* that measures the *error* of the algorithm.

- Experience E involves the algorithm attempting to completing T. Experience, in the colloquial sense of the word, is defined as expertise obtained through the repetition of some given task. The essence of that holds true for machine learning as well: the algorithm gains experience by repeatedly performing T. This paper is only concerned with *supervised learning algorithms*, where the dataset used to train the algorithm specifies the desired output for some given inputs. The dataset used during training is called the *training dataset*, which constitutes E. Each individual output and corresponding inputs being called an *example*; for a tumour classification, as discussed above, the input data would be characteristics like the size of the tumour and its location, and the output would be a binary value representing the two categories.

The simplest machine learning algorithm is linear regression. Linear re-

gression attempts to model the relationship between a set of input and output variables using a linear function. One example use case is for the prediction of housing prices. As evident from the name, the task T is a regression task; in this case it would be predicting the price of a house given some set of features. The experience stream E comes in the form of the training dataset utilised during the learning process, with a performance measure P being used to evaluate its accuracy. For the example of housing prices, the training dataset would specify a house's price and the corresponding features of it. Some cost function would be used to measure P, and for regression tasks the mean-squared error is typically used.

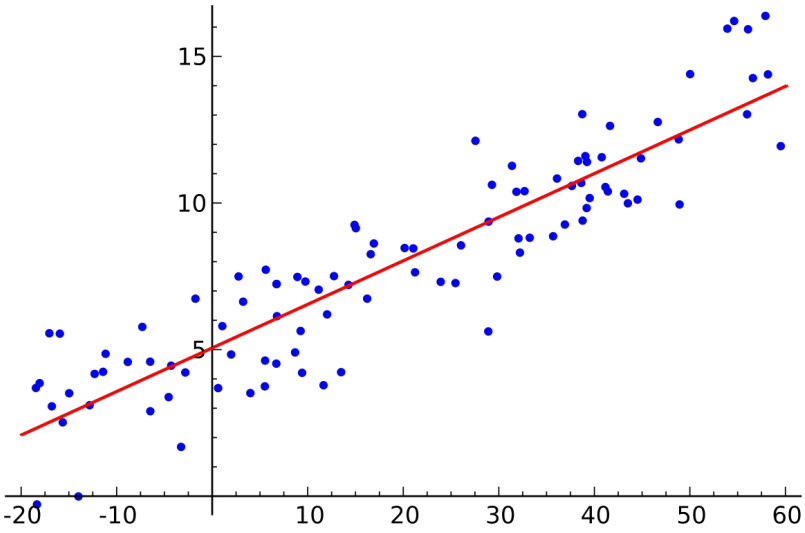


Figure A.1: An example linear regressor (Sewaqu 2010).

Figure A.1 shows an example of a linear regression model. Note how the model is not perfect: this is owing to the fact that the data is *noisy*, or contains some level of randomness. The y-axis would be the output variable, or the target $\hat{y}$, and the x-axis the input examples x. Note that although the output prediction is a scalar value, the input x for a given training example

is typically represented as a vector because there can be more than one input variable for a single output. The model would predict the price of a house not just based on its square footage, but also the number of bathrooms, the number of bedrooms, the existence of a pool, and so forth. For simplicity's sake, Figure A.1 treats x as a scalar value.

There is one key element missing: the learning process. A machine learning model must have some means of adjusting its behaviour to improve at T, otherwise it would not be capable of improving. Recall that a machine learning model attempts to map some set of inputs x to their corresponding outputs y. This mapping is defined using a set of parameters. The learning algorithm iteratively adjusts that set of parameters such that the model is becomes a good mapping from x to y. Note that the quality of the mapping is measured by the performance measure P. The most common iterative learning algorithm is gradient descent, but its precise mechanics are outside the purview of this paper.

A.2 Deep Learning

Deep learning algorithms are just machine learning algorithms that use multiple layers to extract progressively higher level *features*, or characteristics, from the raw input.

A.2.1 Motivation

Previously, this paper discussed linear regression as an example of a machine learning algorithm. More generally, there are a plethora of other machine

learning algorithms, such as k-means clustering (a classification algorithm) and decision trees, which can be used for an appropriate task. Why are deep learning models necessary?

This question can be answered by considering the fundamental problem that machine learning algorithms face with certain tasks: the *curse of dimensionality*.

In statistics, dimensionality refers to how many attributes a dataset has (Glen 2017). In the housing price prediction example, the dataset has relatively low dimensionality because it has relatively few features. This makes simple machine learning algorithms such linear regression appropriate. However, datasets can have millions of features for a given training example, which increases the dimensionality significantly.

Figure A.2: Graphic visualising varying dimensionality. *Left*: Low-dimensional data, with a space of $10^1 = 10$ possibilities. *Center*: Medium-dimensional data, with a space of $10^2 = 100$ possibilities. *Right*: High-dimensional data, with a space of $10^3 = 1000$ possibilities.

Consider Figure A.2. In each of the three regions, the grid represents the total space of possible values the input and output variables could take. The position of a region indicates the possible values the input variable(s) could take, while the color of the region indicates the value of the output. Some region being occupied with a color means there exists a training example

for it. In the left-most diagram in Figure A.2, the majority of possibilities have a training example associated with them. In the middle diagram, the proportion of the whole space of possibilities occupied by training examples is smaller. The same pattern holds for the right-most diagram.

Unfortunately, humans are only capable of visualising three dimensions. However, extending the previous pattern to higher dimensions reveals a general problem: for higher-dimensional data, a smaller space of possibilities is occupied by the training data. Human language is an excellent example. Obviously, of all the possible combinations of letters for a given length, only a few will be coherent sentences. This makes sense, because there are more combinations of letters that result in nonsense than words, especially for longer words.

That therefore poses quite a severe statistical challenge: the number of configurations of x is much larger than the number of training examples. For a low-dimensional dataset, the low number of grid cells are mostly occupied by data, as in the left region in Figure A.2. During prediction, the model usually can just inspect the training examples that lie in the same cell as the new input: if a house in the training set with 4 bedrooms sold for \$500K, another house with 4 bedrooms should also logically sell for \$500K (assuming all other factors held equal)[1]. But what about cells which have no data? In higher-dimensional spaces, the proportion of the space of possibilities occupied by training examples is quite low. Therefore, the average grid cell has no training example associated with it.

The typical solution to this problem is approximating the output at a

1. This analogy actually illustrates the effect of randomness, or *noise*, on a dataset: there may be some factor not included in the dataset, or pure random chance, which accounts for two training examples with the same features being sold for different prices.

new point from the nearest training points. However, this is difficult to do for extremely high-dimensional data, like natural language. The proportion of configurations present is just too small for any meaningful approximation to be made.

Generally, the main limitation behind conventional machine learning algorithms is that as the dimensionality of a dataset increases, the number of examples required increases by $O(v^d)$, for d dimensions and v values. This makes training them on high-dimensionality data infeasible.

Deep learning solves this problem by constructing models that use multiple layers to extract progressively higher level features from the raw input.

A.3 Neural Networks

At the heart of deep learning is the artificial neural network, henceforth abbreviated as ANN and alternatively referred to as neural networks.

Neural networks are essentially an interconnected collection of nodes called artificial neurons, which attempt to model the neurons in a biological brain. Figure A.3 illustrates both the general architecture of a neural network, and how it extracts progressively higher-level information from the raw input data, with the example of image classification. It is difficult for a computer to comprehend raw image data, because the function mapping between a set of pixels and some output class is incredibly complicated, and the curse of dimensionality explored previously makes it impractical to use traditional machine learning algorithms. Deep learning solves this problem breaking down the complicated mapping into a composition of simple mappings. Each of these are described by a layer in the model.

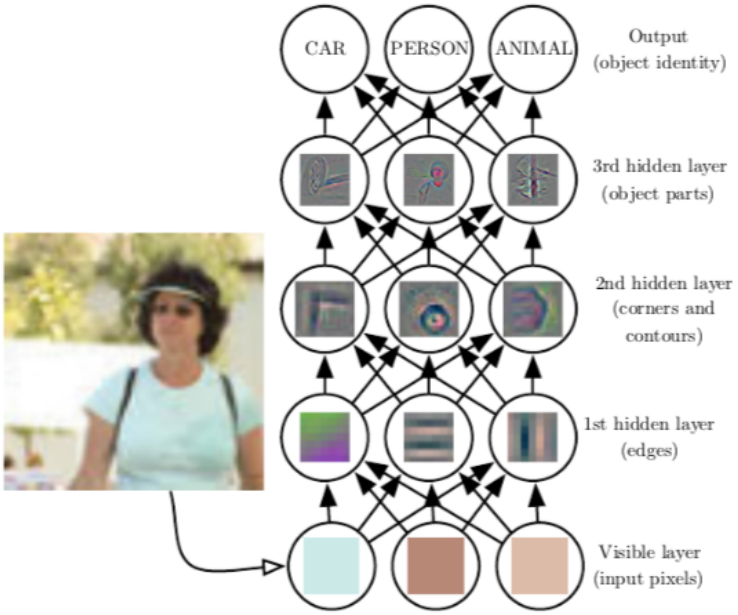


Figure A.3: A diagram illustrating how a neural network uses layers to extract higher-level information than would be possible with only a single layer. The images here are visualisations of the features represented by each hidden unit (Goodfellow 2016).

In Figure A.3, the task is to classify images into some set of identities. Instead of attempting to create a single function which performs the desired mapping, each layer is responsible for extracting progressively higher-level features. The 1st hidden layer detects edges in the raw image. The 2nd hidden layer uses those edges to detect corners and contours. The 3rd hidden layer uses those corners and contours to detect object parts, and output an appropriate identity. The mapping encoded at each hidden layer is relatively simple, for that's all it needs to be. Edge detection, in itself, is not excessively complicated. Neither is the feature extraction at any of the other layers. But by combining them together, the neural network can effectively model a complicated mapping of high-dimensional data.

Neural networks are made up of three different types of layers, as shown in Figure A.3. The first of these is the *input layer*, which constitutes the raw input data. Another type is the output layer, which constitutes the output of the neural network. The last type of layer is the hidden layer, which are the mappings in between the input and output layer not directly visible to the user (hence their name, hidden). Each of these layers is a composition of a non-linear activation and linear function[2].

The learning process for a neural network is similar to that of a traditional machine learning algorithm. A gradient-based iterative approach, such as gradient descent, is still used. However, an algorithm known as *backpropagation* is used to adjust the weights of hidden layers nestled deep within the neural network. The specific mechanics of backpropagation is out of this paper's scope, as it is not necessary to understand the experiment conducted.

2. The use of non-linear activation function is what makes neural networks *universal function approximators*: it has been mathematically proven that neural networks can approximate any mathematical function.

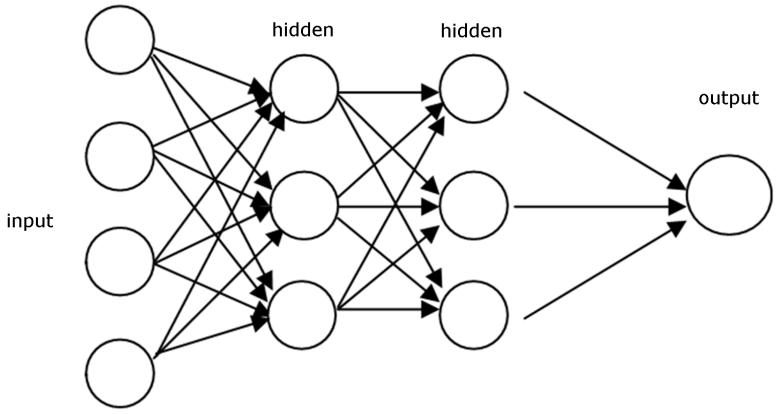


Figure A.4: An example of a feedforward neural network (Packt). Note how information can only flow towards the output layer.

Feedforward Neural Networks It is important to note that the type of neural network architecture being discussed is known as the *feedforward neural network.* They are called feedforward because connections between nodes do not form a *cycle*: the flow of information takes place in the forward direction. The inputs pass from left to right, moving linearly towards the output. This is not necessarily the case with other architectures.

Miscellaneous Terminology There is some miscellaneous terminology referenced throughout the paper that has yet to be defined. *Hyperparamters* refer to parameters which control the architecture of the neural network (for example how many layers it has, or how many nodes per layer) and the behaviour of the learning algorithm (Radhakrishnan 2017). *Overfitting* refers to when the model fails to *generalise*; that is, when the model is unable to properly function on data not contained in the training set (Goodfellow 2016). *Padding* is the practise of adding filler data to the start or end of a

sequence of data to maintain a constant input sequence length (Flach 2017).

Appendix B

BERT and XLnet

The purpose of this chapter is to give an overview of how XLnet and BERT work, as well as explain any background theory necessary.

B.1 NLP

Natural language processing (abbreviated as NLP) is the branch of AI concerned with creating computer systems capable of understanding human language. There are a wide variety of tasks which fall under NLP, ranging from reading comprehension on a given passage, to speech recognition from an audio file, to analysing the sentiment of a text. This paper is concerned with the application of different deep learning models to NLP tasks.

B.1.1 Language Representation

Language representation is the task of representing human language using vectors. Given some set of words x, a model would translate them into a set of representations z. This is extremely difficult, as the model has to learn

how to mathematically represent not just what the word itself is, but also the semantic meaning of the word.

Autoencoding merely refers to the application of autoencoders to language representation. Autoencoders are a special type of neural network. Instead of taking as input a set of words and outputting their representation, an autoencoder will take a corrupted version of a set of words as the input and attempt to reconstruct the original clean version as the output. The language representations themselves are actually encoded in the very parameters of the model. Note that language representations are sometimes called word embeddings.

Models trained to extract language representations have successfully been applied to a variety of other NLP tasks, with BERT being one such example.

B.1.2 Language Modelling

Language modelling is the task of assigning a probability to: (1) sentences in a language (e.g. "what is the probability of seeing the sentence *the man likes eating burgers*?", and (2) the likelihood of a given word, or sequence of words, following a sequence of words ("what is the probability of seeing the word *dog* after the sentence *the quick brown fox jumps over the lazy?"*). XLnet is an example of a model applied to the task of language modelling. The adjective "autoregressive" is sometimes used to describe language modelling.

What is the point of language modelling? The articulation of the language modelling task belies its usefulness. An algorithm with perfect performance on the language modelling task (that is, it is capable of predicting the next word in a sequence with the same number of, or less, guesses than

a human) would have human-level intelligence. This is because almost every NLP task can be expressed in terms of sentence completion. For example, the answering question X could be phrased as a sentence completion task: "the answer to question X is _____."

B.2 Tokens in NLP

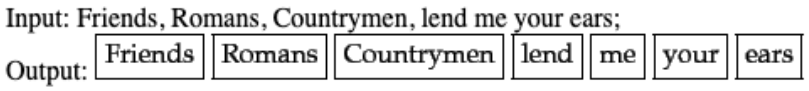


Figure B.1: An example of tokenization.

In NLP, a process called *tokenization* is responsible for splitting text into *tokens* (what we refer to as words) based on whitespace and punctuation (Goldberg 2017). In general, we always tokenize a text before passing it as input to an NLP model, because it makes the input easier for the model to understand.

B.3 BERT

The purpose of this chapter is to explain how BERT works, and the underlying model architecture.

B.3.1 The Transformer

BERT is based almost entirely on a network architecture called the Transformer. It is designed for machine translation (the task of translating from one language to another), and there are several key characteristics of its architecture which are of note.

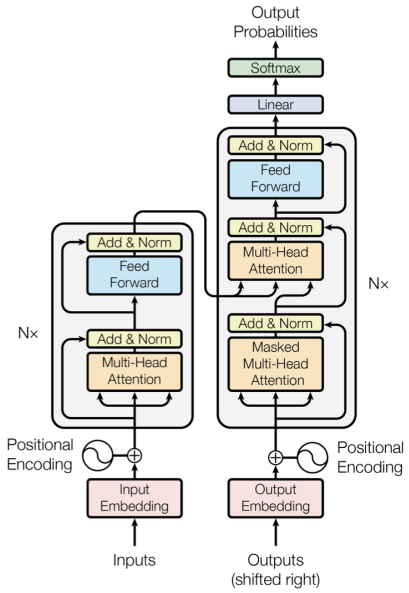


Figure B.2: The Transformer Architecture.

Encoder and Decoder Stacks The Transformer breaks down the task of language modelling into two problems: encoding the input sequence $(x_1, \cdots, x_n)$ into some representation $z = (z_1, \cdots, z_n)$, and decoding the representation

z into some output sequence $(y_1, \cdots, y_n)$ of symbols. It therefore uses an encoder-decoder structure, as shown in Figure B.2. All encoders share the same structure, but have unique sets of parameters. The same holds true for decoders.

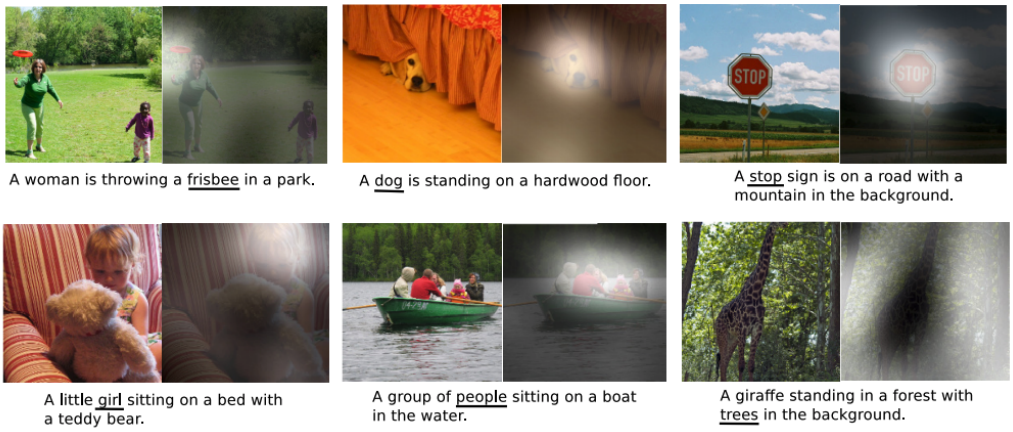


Figure B.3: A visual example of attention, for a model that attempts to generate captions from an image (Xu et al. 2015). *White* indicates an attended region, while *underlines* indicate the corresponding word.

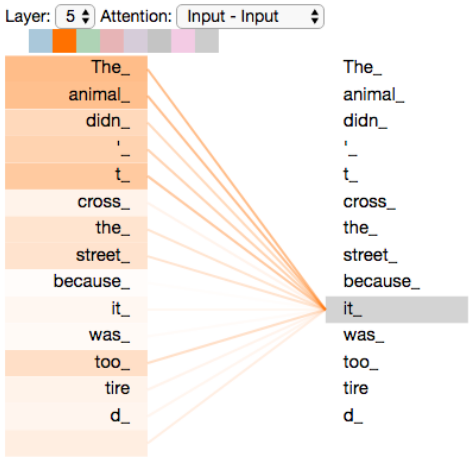


Figure B.4: An example of the self-attention mechanism (Alammar). The intensity of orange highlighting on the left indicates how much attention is being paid to those words.

(Self-)Attention Attention mechanisms allow a neural network to focus on specific parts of the input data when processing it. Intuitively, this makes sense because certain parts of the input are going to be more valuable than the rest. A visual example of attention, albeit in a different context, can be found in Figure B.3. Notice how for a given word, the attention is centered on the corresponding region in the image. This highlights how attention mechanisms allow a network to focus on the relevant parts of the input data. Self-attention is a special type of attention mechanism used by the Transformer. Consider the following example sentence: "The animal didn't cross the street because it was too tired." Self-attention allows the model to, when processing the word "it," associate it with other relevant words in the sequence. A visualisation of this can be found in Figure B.4.

Positional Encoding In order for the model to utilise the order of the sequence, there must be some information injected about the relative or absolute position of the tokens in a sequence. To do so, "positional encodings" are appended to the input at the bottom of the encoder and decoder stacks (that is, before the encoder and decoder process them).

The Transformer obtained state-of-the-art results on the WMT 2014 English-to-German and English-to-French translation tasks. This is likely because it is capable of taking into account the bidirectional context of a token during prediction, and it works with sequences of variable length.

Bidirectional context is very important. Consider the example of a neural network trying to learn the representation of the word "Teddy" from the following two sentences:

1. '*He said, "Teddy* Roosevelt was a great President."'

2. '*He said, "Teddy* bears are on sale!"'

Both of those sentences have the same left-to-right context for the word "Teddy." If the model were to only take into account left-to-right context, it would treat the "Teddy" in both examples as having the same meaning, but their meaning is dramatically different! Its meaning is informed not just by the words that come before it (left-to-right context), but by the words that come after it (right-to-left context). Therefore, it is crucial that a model be able to take into account bidirectional context.

The ability to work with variable sequence length is also very important. The English sentence "I had eggs for breakfast" translated to French is "J'avais des œufs pour le petit déjeuner"; the sentence lengths are 5 and 7 respectively. This illustrates the general point that when it comes to language, the input and output sizes can vary considerably. We would like for a machine translation model to work not just with sentences only of some specific length, but with all sentences.

B.3.2 BERT

BERT stands for Bidirectional Encoder Representations from Transformers. It is, as the authors put it: "conceptually simple and empirically powerful," obtaining state-of-the-art results on 11 different NLP tasks. Its key technical innovation is in applying the Transformer architecture to the task of language representation with a bi-directional context.

Model Architecture BERT is built entirely on the encoder part of the Transformer, so much so that the authors omit a background description of their architecture and instead refer readers to the original Transformer paper. One key characteristic of BERT is its unified architecture across many different tasks: as we'll later see, it is possible to apply BERT to a variety of downstream NLP tasks and obtain state-of-the-art results with only a minor change in architecture.

Pre-training BERT BERT is pre-trained on the two different language modelling objectives: masked language modelling (abbreviated MLM), and next sentence prediction (abbreviated NSP). In deep learning, *pre-trained* refers to the strategy of training a model on some simpler task before confronting the challenge of training on the actual task. In this case, those two objectives are the simpler task, and the actual task will be some downstream NLP task. For MLM, 15% of the tokens in the training data are replaced with a mask token, and during training the model tries to predict what that token is given its bidirectional context. For NSP, the task is what the name suggests: given the sentence pair A and B, the objective is to predict the probability that B follows A.

Fine-tuning BERT Fine-tuning refers to a form of transfer learning. Transfer learning is the practise of taking a model developed for a task and reusing it as the starting point for a second task (Brownlee 2019). The underlying assumption is that training it on the first task (referred to as pre-training) will assist its performance on the second task; the weights of the model are *fine-tuned* for the second task[1]. For each task, we simply plug in the task-specific inputs and outputs into BERT and fine-tune all parameters end-to-end. At the input, sentence A and B are analogous to:

1. Sentence pairs in paraphrasing (summarisation),
2. Question-passage pairs in question-answering, and
3. Text-$\varnothing$ pairs in text classification ($\varnothing$ here just represents a null input: for classifying text into some discrete categories, the model should only require a single sentence)

At the output, there is an additional output layer which serves to specialise the output to the specific downstream task. Token-level tasks[2], where each token in a sequence has an output value associated with it, have a slightly different output layer than segment-level tasks[3], where the sequence of a whole has an output value associated with it. The specific difference in output layer is out of the scope of this paper.

Results BERT obtained 11 state-of-the-art results on various NLP tasks, which empirically demonstrates its powerfulness. These include a 1.5 point

1. The second task is referred to as the *downstream task*.
2. For example, answering a question from a given passage, where the output would indicate which of the words answers the question, is a token-level task.
3. For example, sentiment analysis where it makes sense to look at the sequence as a whole, rather than individual tokens.

absolute improvement on the SQuAD dataset, and a 7.7% improvement on the GLUE benchmark. This proves the potency of pre-training a large model on language modelling and then fine-tuning it with minimal architectural adjustments to some downstream NLP task.

B.4 XLnet

The purpose of this chapter is to explain how XLnet works, and specifically what differentiates it from BERT.

XLnet is a model inspired by BERT's approach to learning language representations. It extends BERT's approach by addressing some perceived flaws, along with integrating some key ideas from Transformer XL, an improved version of the transformer architecture. Because XLnet is not entirely based on Transformer XL, it makes more sense to look at the relevant key characteristics together.

Perceived Flaws with BERT As explored previously, BERT's MLM pre-training objective involves attempting to predict the value of some masked input token. The token to be predicted is indicated, in the input, using the token `[MASK]`. However, there is no such equivalent token during the fine-tuning. This results in a pretrain-finetune discrepancy, as an important element present in the pre-training stage isn't in the fine-tuning stage, thereby hindering performance. Moreover, because the training objective is only to predict some masked input token, it assumes predicted tokens are independent of each other, given the unmasked input. This is overly simplified, as high-order long-range dependencies do exist in natural language.

Autoregressive Objective XLnet has a different training objective from BERT. It is an autoregressive model which, as mentioned in an earlier chapter, means it attempts to predict the probability of a word given context. Because it does not rely on corrupting the input data by masking it, XLnet does not suffer from the pretrain-finetune discrepancy, which allows it to take into account longer-range dependencies.

Attention Although the technical implementation is slightly different, owing to XLnet's different training objective, XLnet uses attention and self-attention in conceptually the same way as BERT.

Recurrence Mechanism XLnet features a recurrence mechanism borrowed from Transformer XL. Its use was motivated by a flaw with the Transformer architecture. The dataset used to train the model, also referred to as the corpus in NLP, had to be segmented and each segment was processed independently. This meant the model ignored all contextual information from previous segments. The recurrence mechanism attempts to address this problem by storing information about a given segment, which is then used as contextual information when processing the next segment.

Results XLnet outperformed BERT on 20 NLP tasks, and achieved state-of-the-art results on 18 tasks, including sentiment analysis and question answering. However, BERT has a plethora of tools available in the repository hosting its code that help with fine-tuning with limited resources. By contrast, the majority of XLnet's state-of-the-art results were obtained on TPUs, which have more RAM than GPUs. This makes it, by the author's own

admission, “very difficult...to re-produce most of [the large XLnet model’s] SOTA results in the paper using GPUs.”