Investigating the effectiveness of Transformers and Performers on SQuAD 2.0

Stanford CS224N Default Project, IID SQuAD Track

Derek Chong
Department of Computer Science
Stanford University
`derekch@stanford.edu`

Abstract

In this project, I explored aspects of the Transformer architecture in the context of question answering on SQuAD 2.0. I hand-implemented and tuned QANet, which significantly outperformed its recurrent predecessor. I then investigated the Performer [1], a Transformer variant, finding it to be viable for use in open-domain QA and scaling the Transformer architecture to large-scale NLP problems. Finally, I designed and built an architectural variant to QANet to support open-domain, open-book question answering, and a TF-IDF-based document retriever which provides open-book "background" context to the former.

1 Introduction

The Transformer [2] is an extremely powerful neural network architecture, which has become the state-of-the-art in several areas of machine learning. In this project, I aim to gain a deeper understanding of its implementation and effectiveness, by applying it to the problem of question answering over SQuAD 2.0, and exploring aspects of the architecture across related subproblems.

I first implement QANet [3] (a Transformer-based Q&A architecture) from scratch, and compare its performance against the provided recurrent baseline architecture, BiDAF [4]. I find that QANet significantly outperforms the baseline, providing a five-point improvement on every scoring metric.

A key limitation of the Transformer architecture is an $O(L^2d)$ scaling on the number of tokens L, of dimensionality d, in the input sequence, due to its self-attention mechanism needing to calculate all pairs of interactions. In practice, this limits L to 512 tokens [5], which precludes the use of Transformers on a wide range of problems. Considerable effort has been spent attempting to overcome this limitation, which has produced a wide range of proposed efficient Transformer models [6]. However, these each come with limitations and tradeoffs, and have generally failed to transfer across implementations and applications [7].

I investigate a recent and highly-promising Transformer variant, the Performer [1], which was published in October 2020 and makes a strong attack on this limitation. This architecture introduces the *Fast Attention Via positive Orthogonal Random features* (FAVOR+) algorithm, which uses kernel methods to overcome the scaling problem in a highly general fashion. The algorithm promises to produce estimates of attention matrices in linear time and space, supporting hundreds of thousands of tokens, and operating at a speed close to not having an attention mechanism at all. It comes with strong theoretical guarantees to be unbiased or nearly-unbiased, and to have good convergence properties and low estimation variance. It is even fully backwards-compatible; a Transformer may be replaced with a Performer in any pretrained model, with no performance impact after only a small amount of fine-tuning. I successfully upgraded QANet to use Performers, and verified that it performs as advertised, scaling linearly to thousands of input tokens while maintaining performance.

Next, I attempt to use this capability to learn how Transformers scale to large inputs, as might be required in open-domain question answering. I extend the QANet design to support a large "background" field, which provides a broader context beyond the 400-word context paragraph. This is supplied by a TF-IDF-based document retriever [8] which searches Wikipedia and returns a page which is relevant to the current context. I delivered a reasonably-performant document retriever component, and completed an end-to-end implementation of the open-domain Q&A system, which appears to train successfully, but was not able to perform the final tuning and training runs due to time constraints.

2 Related Work

While a wide range of Transformer variants exist which attack the above quadratic scaling problem, current consensus seems to be that no dominant solution exists [7] [1] [6]. Broadly speaking:

1. solutions which restrict or simplify the attention calculation process lose rigorous guarantees on representational power and validity,
2. solutions which approximate the attention calculation process impede application to larger-scale problems, and
3. solutions which estimate attention matrices have so far only achieved biased estimators with large error rates, which may not be used on decoder blocks.

A survey of Transformer variants from last September [6] placed the Performers' kernel-based approach at the forefront of current solutions. Additionally, solutions such as Transformer-XL [9] sidestep this problem by reintroducing recurrence.

In my limited review of the literature, I did not find any attempts to apply a Transformer variant to large open-domain question answering, other than a bag-of-words model [10]. This may be due to pre-training becoming the dominant approach to this problem.

The separation of my open-domain question answering solution into document retriever and document reader components was inspired by DrQA [8].

3 Approach

I structured an approach of incremental development, which started with the baseline BiDAF framework and implemented functionality in multiple discrete phases, which each built upon the previous phase. I wrote all code from scratch using each item's original paper as a primary reference (with the exceptions of occasional reference to online materials such as [11] and [12], and some minor convenience functions).

I began by adding character-level embeddings to BiDAF, as this feature was a quick win for an accuracy gain [4] which would carry into the following phases. The baseline code provided convenient interfaces for this purpose. I ran some experiments on the effects of batch size on learning, in order to have confidence in scaling to the latter phases.

Next, I implemented QANet via a process of gradual refactoring of the baseline BiDAF model. Every component in BiDAF has a one-to-one correspondence to a component in QANet, which allowed me to convert the model across part-by-part, while verifying that it continued to function throughout the process. I then improved its performance by conducting hyperparameter optimization plus multiple small experiments in parallel, in order to understand the ranges and effects of key hyperparameters.

I then developed an original version of QANet which used a Performer architecture instead of a vanilla Transformer architecture. This consisted of replacing the self-attention mechanism within each QANet encoder block with the two Performer FAVOR+ attention approximation matrices, and calculating a mapping for Key and Query matrices such that the product of the new matrices correctly approximates the true attention matrix:

The Performer architecture calculates the approximate decomposition of $A = \exp(QK^\top/\sqrt{d})$ into Q' and K' matrices by finding a $\phi(u)$ for $u \in \mathbb{R}^d$, where the following is guaranteed to hold:

$$A(i,j) = K(q_i^\top, k_j^\top) = \mathbb{E}[\phi(q_i^\top)^\top \phi(k_j^\top)]$$

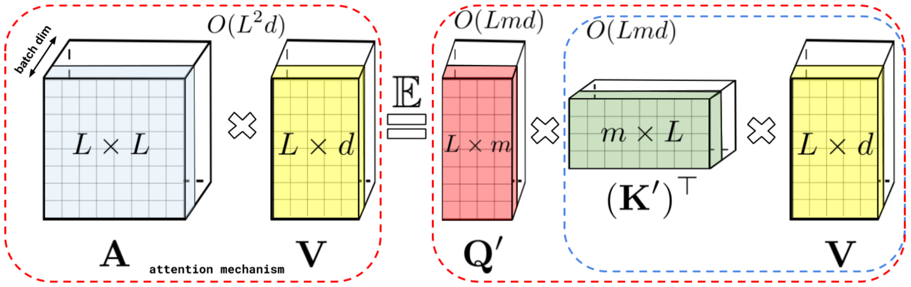

Figure 1: Standard attention module on the left, versus decomposed attention on the right

The individual matrices of $\phi(q_i^\top)$ and $\phi(k_i^\top)$ terms may then be cleanly separated. At a high level, finding the mapping $\phi(u)$ is accomplished by using kernel methods to identify an infinite function which, in expectation, returns the desired term when multipled with its pair (the equivalent of a Taylor series). We can then select the number of features for the infinite series according to our desired approximation accuracy. I compared the two models for performance and scalability.

I then examined the problem of open-domain QA. I decided to focus on open-book QA, which leaves the context available when processing the question, as I found that a large percentage of SQuAD questions depend on the existence of the context passage and are not at all answerable without this (e.g.: *"What is located within this district?"*). This is conceptually similar to a human given a problem with some detail, and being allowed to perform a Google search before responding.

For the document retriever component, I built a basic TF-IDF-based search engine using Gensim [13]. I produced a cleaned corpus of Wikipedia articles, calculated TF-IDF vectors for each article, and ran all SQuAD contexts through this to produce the top search result for each input. Each such article was then extracted, tokenized against the baseline's GloVe word vectors, and mapped to its specific question-context pair.

I then designed an original change to QANet to allow it to use this additional data (*"background"*) while minimizing memory overhead (referred to below as *"QANet-OD"*). I added two elements to its Context-Query Attention layer: Query-Background attention and Context-Background Attention. The first is fed into the Context-Query Attention ($f(q, c) = W_0[q, c, q \odot c]$) as follows:

$$f(q, c, b) = W_0[q, c, q \odot c, q_b(b, c)] \text{ where } q_b(c, b) = W_1[q, b, q \odot b]$$

The second is concatenated to the above Context-Query Attention, such that the final revised Context-Query-Background Attention $f'(q, c, b)$ is:

$$f'(q, c, b) = [f(q, c, b); c_b(c, b)] \text{ where } c_b(c, b) = W_2[c, b, c \odot b]$$

The intention of the above is to provide an additional view of the context and query, which help to situate and further contextualize them within the bigger picture of their domain. For example, a passage which uses the word "security" might receive a better answer if the model is confident about whether this refers to physical security versus psychological security. I implemented this design change and performed some basic testing and tuning.

4 Experiments

4.1 Data

I used the default SQuAD 2.0 dataset. For the open-domain QA section, I also used an Wikipedia dump hosted by the Internet Archive (20 December 2018; 15.5GB), which appears to be in common use by the NLP community [14] [15].

4.2 Evaluation method

Different evaluation methods are required to evaluate the quality of the various phases:

1. BiDAF and QANet: I used the default project evaluation metrics of EM, F1, and AvNA scoring for assessing BiDAF and QANet, in addition to charts of training performance.
2. Performers: I benchmarked Performer QANet against vanilla QANet to compare memory and processing speed scalability, as well as verifying relative accuracy.
3. TF-IDF: I collected a sample of context-background pairs and assessed the relevance of each background to its context.
4. Performer QANet-OD: I was not able to complete performance tuning, and as such provide data on training speed and qualitative performance.

4.3 Experimental details

To increase experimentation bandwidth, I wrote code to scale GPU training to 2-4 GPUs and manage automatic VM provisioning and self-shutdown. I also upgraded the baseline codebase to support for training on (free) Google Colab TPUs. This seemed potentially valuable for this project, as PyTorch can compile non-recurrent structures to XLA for a significant speedup, but only specifically for TPUs. This also provided an additional free and powerful source of training capacity; a v2 TPU provides 180 teraflops of processing power, while the Tesla K80 has 9. Adding TPU support to the default project involved generalizing sections to support the `torch_xla` libraries and multiprocessing idioms, and some memory optimization.

I performed a 3 million-item train run with vanilla BiDAF, to serve as my main baseline. Other runs did not exceed 2 million items irrespective of GPU/TPU count and configuration, as I found that `dev` consistently overfits above this range.

I configured my QANet implementation precisely according to the detailed specifications in the original paper. However, given QANet was designed for SQuAD 1.0 and SQuAD 2.0 is significantly more demanding, I ran a series of manual experiments to determine good ranges for key parameters such as learning rate, batch size and Transformer feed-forward size. I then fed this into a 20-run, 3,000-timestep hyperparameter optimization run using Weight and Biases [16], which provided insight on hyperparameter ranges and relative importances:

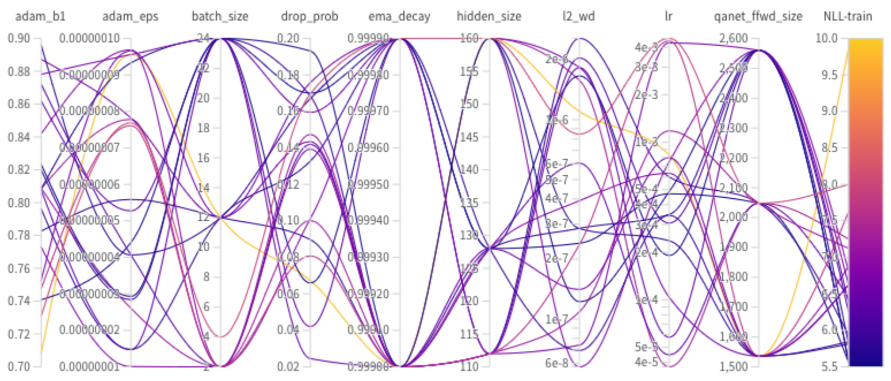

Figure 2: Hyperparameter optimization run

Surprisingly, the only resulting variant which outperformed the paper's hyperparameters on SQuAD 2.0 was changing the learning rate from 0.001 to 0.0003, which is a clear "sweet spot" within the diagram above. I believe a larger per-run training budget with more parameters might have surfaced additional beneficial changes, as candidates tended to perform well initially but lag behind outside the 3,000-timestep mark. I also tested enabling automatic mixed precision and the PyTorch cuDNN autotuner; the former improved training performance on all platforms which supported it.

For my Performer QANet implementation I used 512 for the number of approximation terms m (see Figure 1 above) but otherwise maintained the above parameters, as the replacement approximates the original. I implemented but ultimately chose to disable orthogonal random features. This architecture option provided significant increases in approximation accuracy by using the Gram-Schmidt process to orthonormalize the randomness used within the algorithm, but requires that $d > m$, which significantly limited m given 8-headed attention and $d = 512$. I ran benchmarks against vanilla QANet using the a single GPU, set to a batch size of 1, and padded the input to the lengths under test.

For my TF-IDF implementation I used word vectors of the 100,000 most frequently-occurring words, and maintained the QANet hyperparameters as a starting point.

4.4 Results

4.4.1 Default Metrics

My implementation of BiDAF character-level embeddings and vanilla QANet performed several points better than baseline:

Algorithm	Variant	Stopped at	AvNA	EM	F1
BiDAF	Baseline	1.3m	65.0	55.6	58.6
BiDAF	With character embeddings	1.3m	67.1	58.2	61.0
QANet	Batch size 48	1.4m	69.6	60.4	63.5
QANet	Batch size 32	1.2m	**70.6**	**61.6**	**64.6**

Table 1: Model evaluation metrics

When I ran my full-length QANet train at batch size 48, I found that training loss decreased to an extremely low level (below 1.0), while validation loss remained high. This suggested the model was memorizing inputs. I regularized by decreasing the batch size to the recommended 32 items, and gained an additional point of performance on all score dimensions.

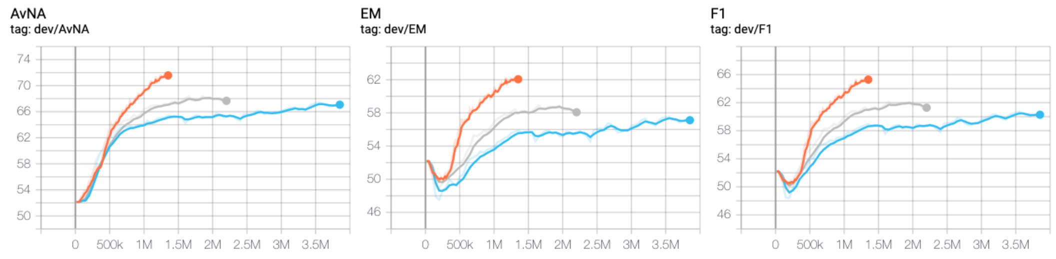

Figure 3: Training performance of BiDAF (blue), BiDAF with char embeddings (grey), QANet (orange)

I submitted the final QANet entry to the Test Leaderboard and received a EM / F1 scores of 60.58 and 64.08 respectively. This result is much lower than the original paper's EM / F1 of 73.6 / 82.7 respectively, but I believe this is ultimately attributable to the increased difficulty of SQuAD 2.0 when compared to SQuAD 1.0.

4.4.2 Performer Performance

I used a debugger to inspect the values output by the self-attention calculation in vanilla QANet and Performer QANet. Using $m = 512$ random features, approximations were consistently within about 5-15% of their true values, which could be improved to about 0-2% at $m = 2048$, or by enabling orthogonal random features. I sanity-checked that performance and accuracy remained roughly consistent between models across several small-scale tests, as shown in Figure 4.

I next verified that as expected, Performer QANet and QANet-OD scale linearly in time and space with input size, while vanilla QANet and QANet-OD scale quadratically (Figure 5). At small scales, all models behave similarly in terms of space usage, but Performers exhibit a small performance penalty which reaches breakeven at about 2,250 elements. In practice, Performer QANet-OD was able to train at about 20 items per second on a 4-GPU VM with an input size of 2,000 elements and input padding disabled.

I was surprised by the excellent space and time efficiency of this model; my initial estimates placed the performance penalty at a 3x multiple of vanilla attention. It turns out that while this is accurate, because QANet-OD does not use the background after the Context-Query Attention stage, at small

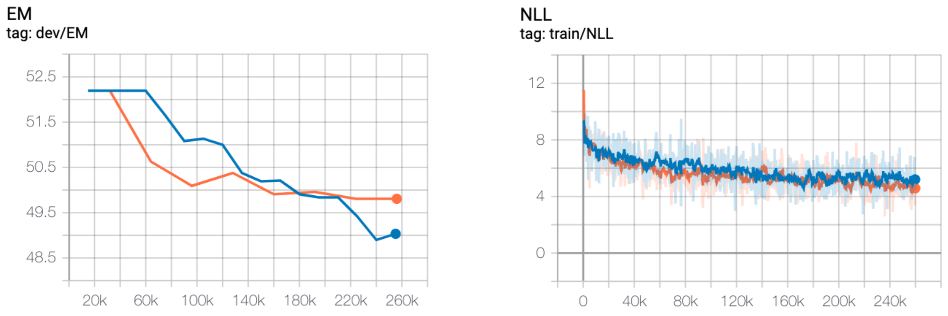

Figure 4: Sanity-checking vanilla (blue) vs. Performer (orange) QANet performance over 2 epochs

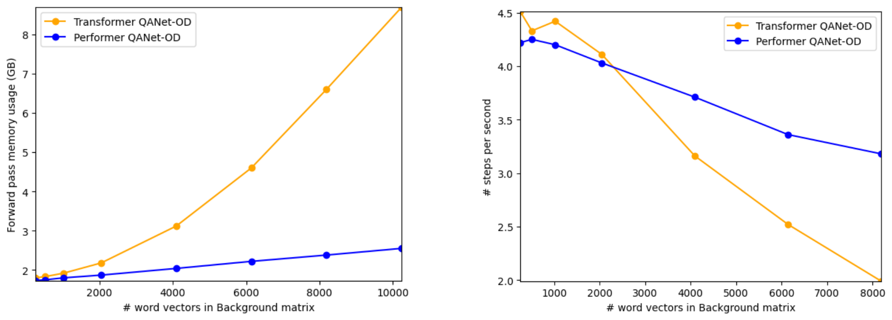

Figure 5: Performers: Memory and performance scaling comparison

scales this penalty is dwarfed by the costs of the overall model, and at large scales vanilla attention's quadratic costs dominate the calculation.

4.4.3 Performer QANet-OD

I ran a few small tests followed by two larger training runs, beginning with similar parameters to vanilla QANet, and trying some variants before reaching our time constraints. I found that both vanilla and Performer QANet-OD quickly found the solution of always returning "N/A", followed by an extended period where both train and test loss declined slowly, while scores stayed constant.

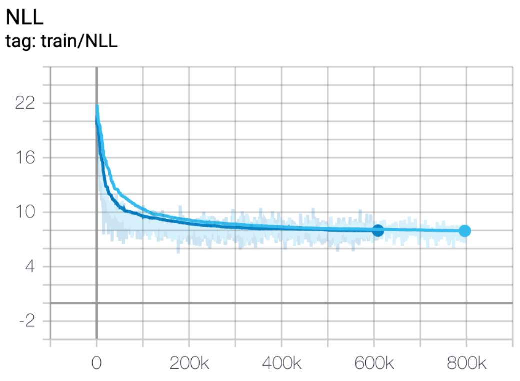

Figure 6: QANet-OD training loss

5 Analysis

5.0.1 QANet

I found that QANet trained extremely slowly, requiring over half a day to see results despite the paper's findings of being "3x to 13x faster [than BiDAF] in training". This is likely due to PyTorch limitations: depthwise separable convolutions are currently slower than regular convolutions in PyTorch [17], and since QANet is completely non-recurrent, its authors may have compiled it with Tensorflow XLA for this performance gain.

5.0.2 Key hyperparameters

I found that any increases in batch size, feed-forward size or hidden size whatsoever tended to result in markedly poorer training performance. I believe this is due to the nature of the SQuAD 2.0 dataset: at a size of 129,941 items, with enough exposure, it is easier for a a large model to improve its training performance via memorization than learning the underlying concepts. Through experiencing significant challenging in training performance, I confirmed that batch size is particularly problematic, confirming that it leads to poorer generalization [18] and sensitivity to choice of learning rate [19]. The cited papers suggest that increasing batch size generally requires a corresponding increase in learning rate in order to maintain training performance, which I found to help in some circumstances.

5.0.3 TF-IDF

In qualitative terms, the search process appeared to be reasonably effective in identifying Wikipedia pages related to a given context passage. I performed an error analysis of inaccurate results, and learned that the primary source of error appears to be that the vector representation for a context paragraph is often dramatically different from the vector for a Wikipedia page as whole. This stands to reason as subsections of Wikipedia pages tend to hold extremely different content from each other. One possible solution might be to divide our corpus into subsections and search over this. However, I believe the current rough approach serves as a sufficiently-good proxy for providing general background information to the model, functioning similarly to a Google search on the topic.

5.0.4 QANet-OD

I believe the training result in Figure 6 above suggests that further hyperparameter tuning may result in the model training correctly - I experienced an identical phenomena with the QANet model when hyperparameters were set suboptimally. Additionally, an increase in training time requirements may come naturally with the large increase in input dataset. In general, given that the model finds the correct starting point and loss consistently decreases, I am positive that with some further effort, this design should produce interesting results.

6 Conclusion

This project has explored a wide range of aspects around question answering and the Transformer architecture, and produced several positive outcomes: (1) my QANet implementation significantly outperforms baseline, (2) the Performer implementation appears to be scalable and effective in the domain of open-domain question answering, (3) TF-IDF was shown to function well as a document retriever, and (4) the novel Performer QANet-OD seems likely to generate positive results.

The key limitation of this project is the remaining open question around whether Performer QANet-OD will work once optimally tuned, and completing this scope is a key avenue for future work.

References

[1] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[3] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

[4] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[5] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[6] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

[7] Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

[8] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[9] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[10] Ivano Lauriola and Alessandro Moschitti. Answer sentence selection using local and global context in transformer models. In *European Conference on Information Retrieval*, 2021.

[11] Min Sang Kim. Implementing qanet (question answering network) with cnns and self attentions. `"https://towardsdatascience.com/implementing-question-answering-networks-with-cnns-5ae5f08e312b"`, 2018.

[12] Ajay Halthor. Depthwise separable convolution - a faster convolution! `"https://www.youtube.com/watch?v=T7o3xvJLuHk"`, 2018.

[13] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[14] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*, 2019.

[15] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[16] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[17] Yun Chen. Pytorch: Fp32 depthwise convolution is slow in gpu. `"https://github.com/pytorch/pytorch/issues/18631"`, 2019.

[18] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[19] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.