Tachyum

Research Paper

Unprecedented Scale and Efficiency in

Generative AI with FP8 8:3 Super-Sparsity

Abstract

The current trend in the field of deep learning is to increase the number of model parameters, so that they can handle difficult tasks and model complex nonlinear systems such as natural language processing. However, it has its downside, as the complex system requirements to train such large models become the prerogative of just a few labs in the world. This paper focuses on the benefits of compression for models both during and after training. Our work shows that it is possible to jointly quantize the model during training in 8-bit accuracy and then perform block pruning, which leaves only 37% of the parameters. This method significantly increases the speed of training and at the same time reduces the memory footprint of the model after training. Moreover, if the hardware on which the model runs effectively supports 8-bit operations and sparse matrices, this further accelerates the model.

Key words

8-bit Floating Point, Quantization Aware Training, Post-training Quantization, Block Pruning, Model Compression

1. Introduction

AI will have a very large impact in the coming decade. Rapid and continuing AI progress is a predictable benefit of the exponential increase in computation used to train AI systems, because research on “scaling laws” demonstrates that more computation leads to general improvements in capabilities.

The three main ingredients leading to predictable improvements in AI performance are training data, computation, and improved algorithms and architectures. The amount of computation going into the largest models was growing at 10 times per year (a doubling time 7 times faster than Moore’s Law). This idea was made precise by developing scaling laws for AI, demonstrating that you could make AI smarter in a predictable way, just by making them larger and training them on more data. Justified in part by these results, this team led the effort to train GPT-3, arguably the first modern “large” language model, with over 173 billion parameters.

Today, the scale of the largest AI computations is doubling every six months, far outpacing Moore's Law. The capacities of language models increase dramatically by more than 1,000 times within a few years, from BERT's 340 million parameters [1] to the Megatron Turing's 530 billion dense parameters [2] and to the sparse Switch Transformer's 1.6 trillion sparse parameters [3] and lower precision (bfloat16). Scaling up language and vision models has been incredibly successful. It significantly improves a model's performance on language and vision tasks, and the models demonstrate amazing few-shot capabilities similar to that of human beings.

There are multiple common approaches for scaling deep learning and for reducing the overall number of computations required for deep learning inference. Quantization is a way of compressing DNN models by reducing the precision of model parameters and/or activations [4], [5]. The immediate benefit of quantization is reduced memory consumption, which allows reduced off-chip storage and bandwidth. In addition, quantization reduces the energy consumption of the matrix multiplication unit via low-bit precision arithmetic. Pruning is a procedure of making DNN models sparse by removing those redundant/insensitive parameters. While it has been observed that having a dense model may be necessary to successfully train a model, it is also possible to remove many of the parameters after the model has been trained without any quality degradation.

Tachyum's Prodigy processor provides support for both dense and block sparse matrix multiplication using the following data formats, float32 as well as via low precision data types: bfloat16, float8 [6], INT8 and INT4 thus allowing to scale deep learning via joint quantization and pruning [7]. It supports block sparsity [8] with compression ratios 4:8 and 3:8.

Our contributions are as follows:

» We propose joint quantization using float8 data type with 8:3 block pruning.

» We demonstrate 8:3 block pruning with quantization aware training (QAT) and adaptive scaling using float8 data type applied to computer vision models and present results.

» We demonstrate 8:3 block post training pruning with post training quantization (PTQ) along with retraining using float8 data type applied to language transformer models and present results.

2. Related Work

Broadly speaking, sparsity can be categorized into two branches: unstructured sparsity and structured sparsity. Unstructured sparsity allows arbitrary patterns of pruning for parameters and feature maps [9]. It can achieve a high compression ratio, but is not hardware friendly. Structured sparsity prunes blocks of subnetworks of a neural network [10], [11], [12].

2.1 Pruning methods

Pruning is a set of techniques for removing weights, filters, neurons, or other structures from a neural network. Pruning can compress standard networks across a variety of tasks, including computer vision and natural language processing, while maintaining the accuracy of the original network. Pruning leads to the reduction of the parameter count and resource demands of neural network inference by decreasing storage requirements, energy consumption, and latency.

Broadly speaking, there are two main approaches for neural network pruning: magnitude-based and impact-based.

Magnitude-based methods use the magnitude of weight to determine its importance and whether or not it should be pruned [13].

Impact-based pruning methods remove weights based on how much their removal would impact the loss function, often using second-order information on the loss function. Impact-based pruning dates back to the work of LeCun et al. [14] where the OBD (Optimal Brain Damage) framework is proposed. This approach, along with subsequent ones make use of second order approximation [15], [7], [16]. Using this approximation, the pruning method minimizes the discrepancy between the output of the original layer and that of the compressed one. Usually, this discrepancy is measured in l2-distance on a small amount of calibration data.

We identify two types of pruning techniques, sparsity aware training prunes the network throughout the standard training process [8], producing a pruned network by the end of training. The other type, post training pruning [17], prunes after the standard training process.

2.2 Post-training pruning

Pre-trained large language transformer models are over-parameterized and difficult to deploy. Therefore, the problem of compressing these models with minimum accuracy loss for downstream tasks is widely explored. Post-training compression has traditionally been investigated in the context of quantization to reduce the computational cost of quantization-aware training. More recently, it has been shown that it is possible to also perform accurate post-training pruning.

Specifically, when parts of the network are removed during the pruning step, accuracy typically decreases. It is therefore standard to retrain the pruned network to recover accuracy [18]. Pruning and retraining can be repeated iteratively until a target sparsity or accuracy threshold is met doing so often results in higher accuracy than pruning in one shot.

3. Method

In this section, we will first introduce the 8-bit floating point format that we used in our experiments. Subsequently, we will describe the pruning method used to remove more than half of the model parameters. In the last part, we present two compression methods based on these techniques.

3.1 Quantization in 8-bit floating point

Our 8-bit floating point format has a 5E2M structure, which means that 1 bit is dedicated to the sign, 5 bits to encode the exponent, and 2 bits to the mantissa. This is a standard format used in several works [19], [6], [20]. When converting from 32-bit floating point format, we use rounding of the last (second) bit of the mantissa, which is rounded according to the third bit of the mantissa of the 32-bit source number. This operation is denoted as $Q_8(\cdot)$ and for tensor T returns its quantized version. As a compensation of the quantization error defined as mean squared error of T and $\bar{T}$, it is necessary to introduce the scaling of the quantized tensor. For the model parameters W, we decided to use the existing statistics-aware weight binning (SAWB) [21] technique. The input tensors X of the operators were scaled using their maximum absolute value. Our scaling has two parts, a constant component, calculated according to the stated rules, and a dynamic component λ, which adapts during training and learns along with the entire model. Each operator (linear, convolutional) has its own dynamic scaling value. The final form of quantization for input tensors is defined as

$$\bar{X} = Q_8\left(\frac{X}{\lambda_X \cdot \max(|X|)}\right) \tag{1}$$

and for model parameters is defined as

$$\bar{W} = Q_8\left(\frac{W}{\lambda_W \cdot \mathrm{SAWB(W)}}\right) \tag{2}$$

$$\bar{b} = Q_8\left(\frac{b}{\lambda_X \cdot \max(|X|) \cdot \lambda_W \cdot \mathrm{SAWB(W)}}\right) \tag{3}$$

as well as for output gradient

$$\overline{\frac{\partial \mathcal{L}}{\partial Y}} = Q_8\left(\frac{\frac{\partial \mathcal{L}}{\partial Y}}{\max\left(\left|\frac{\partial \mathcal{L}}{\partial Y}\right|\right)}\right) \tag{4}$$

This format can be used for both QAT and PTQ. We have tested both cases, and they will be presented in the experimental section. At QAT, operators using 8-bit floating point were implemented in two forms. The first, designated as *semi-floating point 8 implementation* (SFP8), quantizes inputs and parameters only during the forward phase amid the training, the backward phase is always in float32 precision. In this form, the quantized values are rescaled back when calculating the gradients. For example, with the linear operator, a matrix multiplication with 8-bit precision is called in the forward phase, and two matrix multiplications with 32-bit precision in the backward phase. The simplified scheme of forward pass and backward pass is in Fig. 1 and Fig. 2. The second form, called *floating point 8 implementation* (FP8), additionally quantizes the output gradient, and the calculation of the input gradients also takes place in 8-bit precision. If we return to the linear operator example, in this case only 8-bit matrix multiplications are used in both directions. The simplified scheme of described backward pass can be found in Fig. 3.

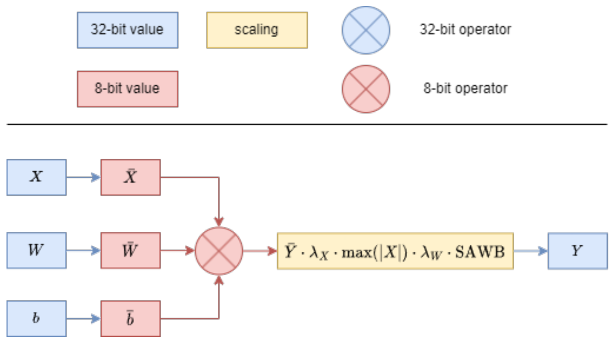

Figure 1
Forward pass with activation *X* and parameters *W* and *b* quantized to 8-bit floating point.

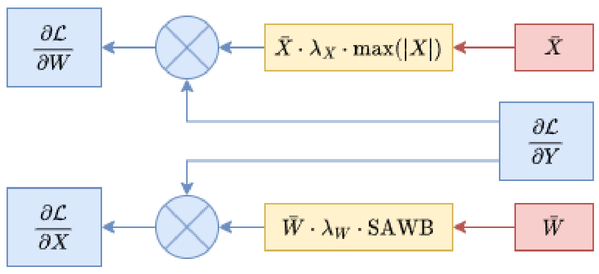
$\frac{\partial \mathcal{L}}{\partial W}$
$\bar{X} \cdot \lambda_X \cdot \max(|X|)$
$\bar{X}$
$\frac{\partial \mathcal{L}}{\partial Y}$
$\frac{\partial \mathcal{L}}{\partial X}$
$\bar{W} \cdot \lambda_W \cdot \mathrm{SAWB}$
$\bar{W}$

Figure 2
Backward pass of SFP8 implementation (in 32-bit floating point precision).

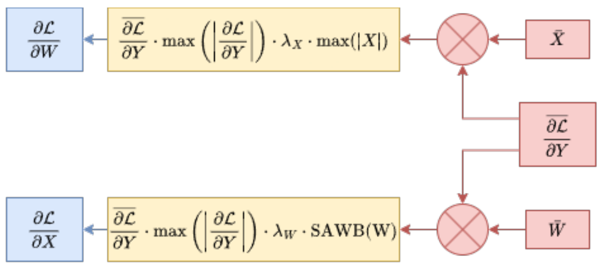
$\frac{\partial \mathcal{L}}{\partial W}$
$\overline{\frac{\partial \mathcal{L}}{\partial Y}} \cdot \max\left(\left|\frac{\partial \mathcal{L}}{\partial Y}\right|\right) \cdot \lambda_X \cdot \max(|X|)$
$\bar{X}$
$\overline{\frac{\partial \mathcal{L}}{\partial Y}}$
$\frac{\partial \mathcal{L}}{\partial X}$
$\overline{\frac{\partial \mathcal{L}}{\partial Y}} \cdot \max\left(\left|\frac{\partial \mathcal{L}}{\partial Y}\right|\right) \cdot \lambda_W \cdot \mathrm{SAWB(W)}$
$\bar{W}$

Figure 3
Backward pass of FP8 implementation (in 8-bit floating point precision).

3.2 Block-wise pruning

For pruning, we use a method that accesses the tensor in blocks and removes parameters based on their magnitude. The input parameters of such a pruning method are the size of the block n and the number of elements k to be preserved. In this way, the parametric tensor is ”cut” into blocks (if necessary, zeros are added to its end so that the total number of elements is divisible by the size of the block) and the k largest elements from each block are kept, the others are set to zero. According to the ratio of n to k, it is possible to create different pruning format. Pruning was always applied after training and quantizing of the model. Subsequently, we always had to retrain the pruned model for a few epochs. The parameters that were already pruned were masked so that they would not be trained. We pruned only Linear and Convolutional layers.

4. Experiment Results

We divided our experiments into three parts. In the first part, we tried to find the lowest possible ratio of n and k for block-wise pruning, for which we used a small visual model trained on a classification task. In the second part, we selected representative models for the classification task, semantic segmentation and detection and applied the same procedure to them as in the first part, but only for the selected ratio n to k. In the third part, we took the trained language model, performed quantization, pruning (again for selected n and k) and fine-tuning. In the first two cases, we trained our own baseline models, so they may not have SoTA performance, but foremost we were interested in detecting a drop in accuracy relative to the baseline models.

4.1 Choosing pruning format

To determine the best possible ratio for pruning, we used the ResNet20 [22] network and the CIFAR10 dataset [23]. We trained the baseline model for 120 epochs to Top1 accuracy 86.95%. Next, the SFP8 and FP8 implementation of ResNet20 was trained, and then pruning and subsequent retraining for 10 epochs was performed on it. The goal was to find such a ratio of n to k, where there was no significant reduction in the accuracy of the model and at the same time it was possible to take the largest possible number of weights according to the rule described in the previous section. We compared five possible formats: 8 to 4, 8 to 3, 8 to 2, 4 to 2, 4 to 1. From the results in Tab. 1 it can be seen that for both implementations of quantization, the best results were achieved for a ratio of 8 to 3, when we keep 37% of the parameters, but the loss in Top1 accuracy is around 1%. The results also show that the network was no longer able to recover from pruning at a ratio of 4 to 1, when only 25% of non-zero parameters remained, while at ratios of 8 to 4 and 4 to 2 still maintained good accuracy with 50% of parameters remaining.

Ratio	Reduction	SFP8	FP8
8:4	50%	0.7	-0.57
8:3	**63%**	**-0.65**	**-1.21**
8:2	75%	-3.06	-3.61
4:2	50%	-0.64	-0.97
4:1	75%	-3.83	-3.92

Table 1
Drop in Top-1 accuracy (%) to baseline 32-bit ResNet20 and reduction of parameters (%)

4.2 Pruning of QAT models

After choosing a ratio of 8 to 3 based on previous experiments, we selected several models that solve different computer vision tasks. Models were trained again in both implementations (SFP8 and FP8). Subsequently, the trained models were pruned and retrained for 10 epochs. For image classification, we chose ResNet32 [22] trained for 200 epochs on the CIFAR100 dataset, ResNet34 [22] and SWIN transformer [24], both trained for 120 epochs on the Imagenet dataset. For segmentation, it was the UNet [25] network and the Kits19 dataset, which we also trained for 120 epochs. Finally, we chose SSD [26] for detection, where the pre-trained ResNet34 served as the backbone. We trained SSD for 240 epochs on the COCO 300×300 dataset. We used Adam [27] optimizer for training and cosine scheduler with warm restarts [28]. A complete overview of hyperparameters is presented in Tab. 2.

Model	Epochs	Warm-up	LR	WD
ResNet32	200	2	3E-03	5E-04
ResNet34	**120**	**2**	**1E-03**	**1E-04**
SWIN	120	5	5E-04	5E-02
UNet	120	0	1E-03	1E-04
SSD	240	0	1E-04	1E-04

Table 2
Hyperparameters used for training SFP8 and FP8 models in our experiments

Based on the results shown in Tab. 3 we can state that pruning format 8 to 3 in connection with quantization to 8-bit float precision can still preserve the performance of the models. Thanks to training in low accuracy and retraining the model after pruning, it turns out that it is possible to save 63% of memory and speed up inference (and also model training) theoretically to 4 times the speed compared to 32-bit accuracy. In the case of classification tasks, there was a loss in accuracy at the level of 1 to 2%, in the case of segmentation there was even an increase in IoU metric, indicating that the model has improved generalization after QAT and pruning. In the case of the SSD detector, there was a decrease of 0.01 in the IoU metric.

4.3 Pruning of PTQ models

To also evaluate our pruning method with 8-bit precision on other domain than vision, we tried natural language processing with question answering task, however this time in a slightly different workflow. To save time and computational resources we did not train the model from scratch, but downloaded a pretrained BERT transformer [1] from HuggingFace Hub [29] with 110M parameters, which was first trained in a self-supervised fashion on English Wikipedia and BookCorpus dataset and later fine-tuned for question answering on SQuAD v1 dataset [30].

Model	Metric	baseline	SFP8	FP8
ResNet32	Top1 (%)	60.06	59.02	58.3
ResNet34	**Top1 (%)**	**68.77**	**66.61**	**66.3**
SWIN	Top1 (%)	75.86	73.41	73.05
UNet	IoU	0.75	0.79	0.79
SSD	IoU	0.32	0.31	0.31

Table 3
Results of models after QAT in SFP8 or FP8 implementation and pruned in 8:3 format after training

After loading the pretrained model, all of the model layers with the exception of the embedding layer were pruned and the model was retrained once again on SQuAD. We have tried several configurations, but the best results were achieved with retraining for just 3 epochs with AdamW optimizer with learning rate: 5e−5, betas: 0.9, 0.999 and weight decay: 0.01. Finally, the model weights from all linear layers but the last one (output layer) were quantized into the FP8 format mentioned previously. The performance of the original model and our model with sparse and quantized weights are evaluated in Table 4. The quality of our PTQ model degrades relative to the original model only from 1.9% to 3.6%, depending on the metric in question.

Metric	baseline	FP8
exact match	80.908	78.004
f1 score	88.228	86.619

Table 4
Results of BERT on question answering task before and after 8:3 block pruning with FP8 quantization

This approach utilizes the theoretical speedup of sparsity during both retraining and inference, but the speedup of 8-bit floats only during inference. When we tried quantizing the model before retraining, the model quality deteriorated rapidly. 8-bit training with the combination of NLP tasks will be the focus of our future research.

5. Conclusion

In our work, we focused on investigating the possibilities of joint application of quantization in 8-bit float precision and block pruning, which preserves less than 50% of the model parameters. Such a compressed model could subsequently use the advantages of the latest hardware platforms, which offer instructions for working with 8-bit floating point numbers and have support for accelerated multiplication of sparse matrices.

On a small ResNet20 model trained on CIFAR10, we experimentally found out the appropriate ratio of block size and number of retained parameters, which preserves the performance of a model. We concluded that 8 to 3 pruning format is a reasonable choice. In the experiments, we then focused on two main types of quantization: QAT and PTQ. The advantage of QAT is that the models are already quantized during training, which speeds up the training process itself. After finishing, the models were pruned and subsequently retrained, which ensured a minimal loss of performance. A model compressed in this way is not only trained faster, but the speed of inference raises as well.

To verify the robustness of our approach, we selected several models designed for visual tasks such as image classification, semantic segmentation, and detection. We applied our compression procedure to them and then compared them with baseline models trained in 32-bit precision and with the full number of parameters. The decrease in performance of the models was still at an acceptable level (there was even an increase in the case of semantic segmentation), if we took into account how much compression was performed on them.

In the second part of the experiments, we looked at large language models, where it is no longer easy to perform their training in a quantized form. That's why we applied PTQ to them in 8- bit precision and then performed pruning and fine-tuning again. We proceeded similarly to the first series of experiments and after PTQ and pruning we compared the resulting performance of the models and could conclude that there was a negligible decrease in performance.

We managed to show that with the increase of current deep learning models, the 8-bit float format will become the standard, and it will be possible to quantize the models in it not only after training, but already during the training itself. In addition, it will be possible to prune the trained model below 50% of the original size without losing performance. These compression approaches, together with hardware support, will make it possible to train and use large deep learning models even for smaller users in private sector or academia.

References

1. DEVLIN J, CHANG M, LEE K, et al. BERT: pre-training of deep bidirectional transformers for language understanding[C/OL]//BURSTEIN J, DORAN C, SOLORIO T. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). Association for Computational Linguistics, 2019: 4171-4186. https://doi.org/10.1 8653/v1/n19-1423.
2. SHOEYBI M, PATWARY M, PURI R, et al. Megatron-lm: Training multi-billion parameter language models using model parallelism[J]. arXiv preprint arXiv:1909.08053, 2019.
3. WILLIAM FEDUS N S, Barret Zoph. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity.[J]. arXiv preprint arXiv:1909.08053, 2022.
4. XIAO G, LIN J, SEZNEC M, et al. Smoothquant: Accurate and efficient post-training quantization for large language models[J]. arXiv preprint arXiv:2211.10438, 2022.
5. YAO Z, YAZDANI AMINABADI R, ZHANG M, et al. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers [J]. Advances in Neural Information Processing Systems, 2022, 35: 27168-27183.
6. MICIKEVICIUS P, STOSIC D, BURGESS N, et al. Fp8 formats for deep learning[J]. arXiv preprint arXiv:2209.05433, 2022.
7. ELIAS FRANTAR D A. Sparsegpt: Massive language models can be accurately pruned in one-shot[J]. arXiv preprint arXiv:2004.14340, 2023.
8. ZHOU A, MA Y, ZHU J, et al. Learning n: M fine-grained structured sparse neural networks from scratch[J]. arXiv preprint arXiv:2102.04010, 2021.
9. MAO Y, WANG Y, WU C, et al. Ladabert: Lightweight adaptation of bert through hybrid model compression[J]. arXiv preprint arXiv:2004.04124, 2020.
10. WANG Z, WOHLWEND J, LEI T. Structured pruning of large language models[C/OL]//Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Online: Association for Computational Linguistics, 2020: 6151-6162. https://aclanthology.org/2020.emnlp-main.496. DOI: 10.18653/v1/2020.emnlp-main.496.
11. YOUNG JIN KIM H H A. Fastformers: Highly efficient transformer models for natural language understanding.[J]. arXiv preprint arXiv:2010.13382, 2020. [12]
12. J.S. MCCARLEY A S, Rishav Chakravarti. Structured pruning of a bert- based question answering model.[J]. arXiv preprint arXiv:1910.06360, 2021.
13. GORDON M A, DUH K, ANDREWS N. Compressing bert: Study- ing the effects of weight pruning on transfer learning[J]. arXiv preprint arXiv:2002.08307, 2020.
14. LECUN Y, DENKER J, SOLLA S. Optimal brain damage[J]. Advances in neural information processing systems, 1989, 2.
15. SINGH D, S. P. Alistarh. Woodfisher: Efficient second order approxima- tion for neural network compression[J]. arXiv preprint arXiv:2004.14340, 2020.

16. FRANTAR E, KURTIC E, ALISTARH D. M-fac: Efficient matrix-free approximations of second-order information[J]. Advances in Neural In- formation Processing Systems, 2021, 34: 14873-14886.
17. WOOSUK KWON M W M, Sehoon Kim. A fast post-training pruning framework for transformers.[J]. arXiv preprint arXiv:2204.09656, 2022.
18. HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network[J]. Advances in neural information processing systems, 2015, 28.
19. WANG N, CHOI J, BRAND D, et al. Training deep neural networks with 8-bit floating point numbers[J]. Advances in neural information processing systems, 2018, 31.
20. MELLEMPUDI N, SRINIVASAN S, DAS D, et al. Mixed precision train- ing with 8-bit floating point[J]. arXiv preprint arXiv:1905.12334, 2019.
21. CHOI J, CHUANG P I J, WANG Z, et al. Bridging the accuracy gap for 2- bit quantized neural networks (qnn)[J]. arXiv preprint arXiv:1807.06964, 2018.
22. HE K, ZHANG X, REN S, et al. Deep residual learning for image recogni- tion[C]//Proceedings of the IEEE conference on computer vision and pat- tern recognition. 2016: 770-778.
23. KRIZHEVSKY A, HINTON G, et al. Learning multiple layers of features from tiny images[J]. 2009.
24. LIU Z, LIN Y, CAO Y, et al. Swin transformer: Hierarchical vision trans- former using shifted windows[C]// Proceedings of the IEEE/CVF interna- tional conference on computer vision. 2021: 10012-10022.
25. RONNEBERGER O, FISCHER P, BROX T. U-net: Convolutional net- works for biomedical image segmentation[C]//Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer, 2015: 234-241.
26. LIU W, ANGUELOV D, ERHAN D, et al. Ssd: Single shot multibox detector[C]//Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. Springer, 2016: 21-37.
27. KINGMA D P, BA J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.
28. LOSHCHILOV I, HUTTER F. Sgdr: Stochastic gradient descent with warm restarts[J]. arXiv preprint arXiv:1608.03983, 2016.
29. WOLF T, DEBUT L, SANH V, et al. Transformers: State-of-the-art nat- ural language processing[C/OL]//Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstra- tions. Online: Association for Computational Linguistics, 2020: 38-45. https://www.aclweb.org/anthology/2020.emnlp-demos.6.
30. RAJPURKAR P, ZHANG J, LOPYREV K, et al. SQuAD: 100,000+ questions for machine comprehension of text[C/OL]//Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. Austin, Texas: Association for Computational Linguistics, 2016: 2383- 2392. https://aclanthology.org/D16-1264. DOI: 10.18653/v1/D16-1264

Tachyum

www.tachyum.com

Tachyum Inc., 8275 South Eastern Ave, Ste 233, Las Vegas, NV 89123, U.S.A.
Tachyum s.r.o., Karadžičova 14, CBC IV., 3rd floor, 821 08 Bratislava, Slovakia

FP8 v1.0_230822