Supplementary material for: Enhancing Transformers Through Conditioned Embedded Tokens

Hemanth Saratchandran
Australian Institute for Machine Learning
Adelaide University
hemanth.saratchandran@adelaide.edu.au

Simon Lucey
Australian Institute for Machine Learning
Adelaide University
simon.lucey@adelaide.edu.au

A. Theoretical Analysis

In this section we give the proofs of Proposition 4.2, Theorem 4.4 and Theorem 4.5.

Proof of Proposition 4.2. We use the well known formula that given two matrices A and B such that AB has full rank then

$$\kappa(AB) \leq \kappa(A) \cdot \kappa(B) \tag{1}$$

where we remind the reader that κ denotes the condition number.

By definition we have that

$$\mathbf{LA}(X) = XW_Q W_K^T X^T X W_V. \tag{2}$$

Applying (1) iteratively we see that

$$\begin{aligned}\kappa(\mathbf{LA}(X)) &= \kappa(XW_Q W_K^T X^T X W_V) \\ &\leq \kappa(W_Q)\kappa(W_K)\kappa(W_V)\kappa(X)^3\end{aligned}$$

where we have used the fact that $\kappa(X) = \kappa(X^T)$.

In the case of softmax self-attention we have that

$$\begin{aligned}\kappa(\mathbf{A}(X)) &= \kappa(\mathbf{softmax}(XW_Q W_K^T X^T) X W_V) \\ &\leq \kappa(\mathbf{softmax}(XW_Q W_K^T X^T))\kappa(W_V)\kappa(X)\end{aligned}$$

which finishes the proof of the proposition. □

Proof of Theorem 4.4. We start be decomposing the embedded tokens into its singular value decomposition (SVD)

$$X = USV^T \tag{3}$$

where U is the $N \times N$ matrix of left singular vectors, V is the $d \times d$ matrix of right singular vectors and S is the $N \times d$ matrix of singular values on the main diagonal. We denote the singular values by $\sigma_1 \geq \cdots \geq \sigma_k$ where $k = \min(N, d)$. We then define a new matrix C by

$$C = U\widetilde{S}V^T \tag{4}$$

where $\widetilde{S}$ is the $N \times d$ matrix that is zero of its main diagonal and on the main diagonal its (l, l) entry is σ_1 for $1 \leq l \leq k$ where $k = \min(N, d)$. If we then add C to X we get

$$X + C = USV^T + U\widetilde{S}V^T = U(S + \widetilde{S})V^T. \tag{5}$$

Note that the matrix $S + \widetilde{S}$ is a $N \times d$ matrix that is zero everywhere except on its main $k \times k$ diagonal where its (l, l) entry is $\sigma_1 + \sigma_l$ for $1 \leq l \leq k$. In particular, if we then compute the condition number of $X + C$ we find that

$$\kappa(X + C) = \frac{2\sigma_1}{\sigma_1 + \sigma_k} \leq 2. \tag{6}$$

□

Proof of Theorem 4.5. The proof of this theorem follows from the result of theorem 4.4. By definition we have that

$$\mu(\mathbf{LA}(X)) = \kappa(W_Q)\kappa(W_K)\kappa(W_V)\kappa(X)^3. \tag{7}$$

From theorem 4.4 we have that

$$\kappa(X + C)^3 \leq \kappa(X)^3 \tag{8}$$

and thus the result follows by plugging this inequality into (7).

For the case of $\mu(\mathbf{A}(X))$ we follow a similar approach. We have by definition that

$$\mu(\mathbf{A}(X)) = \kappa(\mathbf{softmax}(XW_QW_k^TX^T))\kappa(X)\kappa(W_V). \tag{9}$$

The inserting the result of theorem 4.4 that $\kappa(X + C) \leq \kappa(X)$ and using the assumption on $\kappa(\mathbf{softmax}(XW_QW_k^TX^T))$ the result follows. □

B. Experiments

B.1. Implementation of Conditioned Embedded Tokens

In this section we show the reader how we implemented conditioned embedded tokens for all the applications carried out in Sec. 5 of the main paper.

Disadvantage of theory: In Appendix A we gave the proofs of Theorem 4.4 and Theorem 4.5 from the main paper. In the course of the proof of Theorem 4.4 we gave an explicit construction of the correction matrix C. The proof used the SVD of the embedded token matrix X to compute C. One of the problems with implementing this approach is that the SVD computation on a computer will increase time and memory for large scale matrices. Therefore in this section we will give an implementation friendly construction of the correction matrix that we used in Sec. 5 of the paper and that worked well for all experiments.

Implementation for experiments: For the experiments in Sec. 5 of the paper we chose the correction term to be of the form $\lambda \cdot I_k$ where if X is a $N \times d$ and $k = \min(N, d)$ matrix then I_k is the $N \times d$ matrix with zeroes everywhere except for the main $k \times k$ diagonal where it is 1. We will now show that this can be used as an approximation to bring down the condition number of X by taking $X + \lambda \cdot I_k$.

Theorem B.1. *Let X be a fixed $N \times d$ matrix and let $k = \min(N, d)$. Let I_k denote the $N \times d$ matrix with zeroes everywhere except for the main $k \times k$ diagonal where it is 1. Assume that the minimal singular value $\sigma_k(X) << 1$ (much less than 1). Then for $\lambda > 0$ large enough we have that $\kappa(X + \lambda \cdot I_k) \leq 2$.*

Proof. Let us label all singular values of X by $\sigma_1(X) \geq \cdots \geq \sigma_k(X)$. Let $\sigma_i(\lambda \cdot I_k)$ denote the ith singular value of the matrix $\lambda \cdot I_k$. Note that from the structure of $\lambda \cdot I_k$ we have that $\sigma_i(\lambda \cdot I_k) = \lambda$ for $1 \leq i \leq k$. We will then need the following Weyl inequalities on the singular value of a sum of matrices:

$$\sigma_1(X + \lambda \cdot I_k) \leq \sigma_1(X) + \sigma_1(\lambda \cdot I_k). \tag{10}$$

$$\sigma_k(X + \lambda \cdot I_k) \geq \sigma_1(\lambda \cdot I_k) - \sigma_k(X). \tag{11}$$

We then compute

$$\begin{aligned}
\kappa(X + \lambda \cdot I_k) &:= \frac{\sigma_1(X + \lambda \cdot I_k)}{\sigma_k(X + \lambda \cdot I_k)} && (12)\\
&\leq \frac{\sigma_1(X) + \sigma_1(\lambda \cdot I_k)}{\sigma_1(\lambda \cdot I_k) - \sigma_k(X)} && (13)\\
&\approx \frac{\sigma_1(X) + \sigma_1(\lambda \cdot I_k)}{\lambda} && (14)\\
&= \frac{\sigma_1(X) + \lambda}{\lambda} && (15)\\
&= \frac{\sigma_1(X)}{\lambda} && (16)
\end{aligned}$$

where to get the second inequality (13) we have used (10) and (11) and to get the approximation (14) we have used the assumption that $\sigma_k(X) << 1$.

Then observe that if λ is large enough we have that $\frac{\sigma_1(X)}{\lambda} \leq 2$ as required. $\square$

We see that Theorem B.1 gives a nice way to form a correction term for X that does not involve computing the SVD of X. However, it does require the assumption that $\sigma_k(X) << 1$. Empirically we found this assumption holds in all the applications we considered. Fig. 1 shows the full singular value spectrum of the original embedded tokens (red curve) used in the ViT-B architecture on ImageNet-1k verses the singular value spectrum of conditioned embedded tokens, defined by adding a correction term of the form $10 \cdot I_k$ to the embedded tokens. We clearly see form the figure that the embedded tokens has a minimum singular value that is much less than 1.

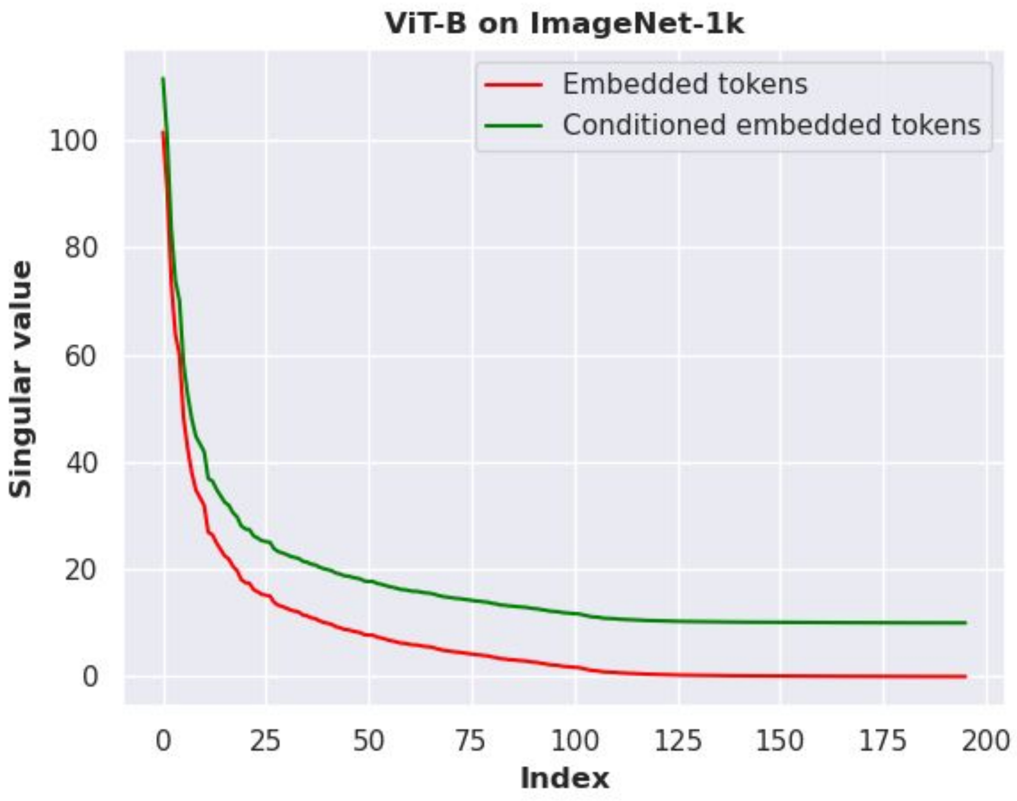

Figure 1. The singular value spectrum of the embedded tokens (red) and conditioned embedded tokens (green) of a ViT-B on the ImageNet-1k dataset. As can be seen from the figure the conditioned embedded tokens has a much higher minimum singular value making its condition number much smaller.

How to choose λ: In the above theory we showed that a correction term can be give by taking the matrix $\lambda \cdot I_k$. This raises the question on how to choose the constant λ. In general, we ran a grid search for each application and found that $1 \geq \lambda \leq 20$ worked the best. Fig. 2 shows how different λ scales affects the accuracy of a ViT-B trained on ImageNet-1k using conditioned embedded tokens with a correction term given by $\lambda \cdot I_k$. As can be seen from the figure we found the best value to be around 10. This pattern followed for all other experiments as well and hence for each experiment in Sec. 5 of the main paper we used a λ scale of 10.

Positional encoding. Our theoretical framework in Sec. 4 of the main paper was carried out without positional encoding for ease of exposition. For the experiments we note that we add conditioned embedded tokens directly after positional encoding and our theoretical framework remains the same.

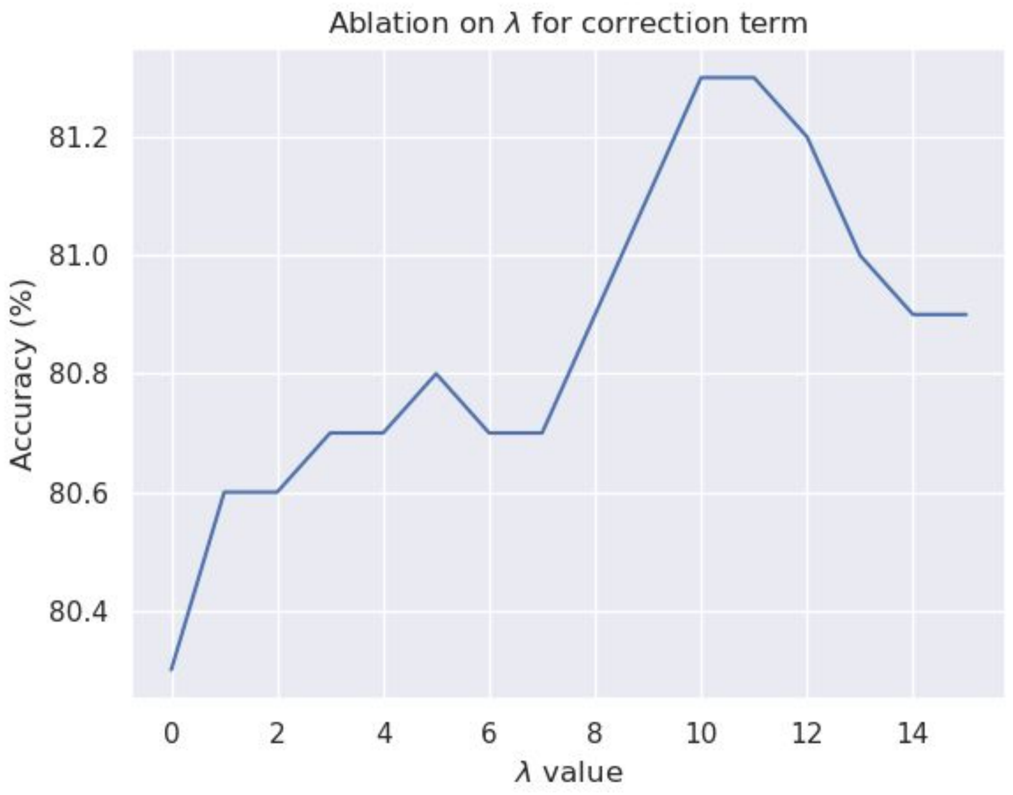

Figure 2. Final accuracy of a ViT-B trained with conditioned embedded tokens with different λ scales on the ImageNet-1k dataset.

Memory and Overheads: In the implementation of conditioned embedded tokens the correction matrix C is fixed during the start of training and thus does not add to the backward pass during optimization. Thus the only additional memory comes from storing the correction matrix $\lambda \cdot I_k$. Since we trained our architectures in pytorch using torch.float32 we find that the memory used to store C when X is $N \times d$ is given by $N \cdot d \cdot 4$ bytes. Therefore, since the embedded tokens are fed into the transformer in batches with a batch size of B, we have that the total memory overhead is given by $B \cdot N \cdot d$. Since the correction matrix is added to the embedded tokens X the total increase in FLOPs is negligible given by $b \cdot N \cdot d$, where b is the batch size.

B.2. Image Classification

Hardware and Implementation: The image classification experiments in Sec. 5.1 of the paper were done on Nvidia A100 GPUs. The implementation of the ViTs were all done using the Timm code base [6]. The architectures were all trained from scratch on the ImageNet-1k dataset using the AdamW optimizer. Since we are training on ImageNet-1k we used a patch size of 16 as each image has height x width = 224 x 224. This implies the sequence length $N = 197$ (with the addition of the class token). The experimental hyperparameters used for each vision transformer is given in Tabs. 1, 2, 3, 4, 5.

Memory and Overheads: For the image classification experiments the sequence length $N = 197$ and the embedding dimension $d = 768$ and the batch size was $B = 1024$. This means the correction term gives an approximate additional memory of 0.5 GB.

Parameter	**Value**
batch size	1024
epoch	300
learning rate	3.00E-03
optimizer	adamw
weight decay	0.3
label smoothing	0.1
warmup epoch	20
warmup learning rate	1.00E-05
mixup	0.8
cutmix	1
drop path	0.1
rand aug	9 0.5

Table 1. Hyperparameter Settings for ViT-B model

Parameter	**Value**
batch size	1024
epoch	300
learning rate	1.00E-03
optimizer	adamw
weight decay	0.05
label smoothing	0.1
warmup epoch	5
warmup learning rate	1.00E-05
mixup	0.8
cutmix	1
drop path	0.1
rand aug	9 0.5
augmentation repetitions	3
random erase prob	0.25

Table 2. Hyperparameter Settings for Deit-B model

Parameter	**Value**
batch size	512
epoch	300
learning rate	1.00E-03
optimizer	adamw
weight decay	0.05
warmup epoch	20
warmup learning rate	1.00E-05
mixup	0.8
cutmix	1
drop path	0.5
rand aug	9 0.5
clip gradient	1
random erase prob	0.25

Table 3. Hyperparameter Settings for Swin-B model

Parameter	**Value**
batch size	512
epoch	400
learning rate	3.00E-04
optimizer	adamw
weight decay	0.3
warmup epoch	20
warmup learning rate	1.00E-05
label smoothing	0.1
mixup	0.8
cutmix	1
drop path	0.1
rand aug	9 0.5

Table 4. Hyperparameter Settings for XCiT-M model

Parameter	**Value**
batch size	512
epoch	300
learning rate	3.00E-04
optimizer	adamw
weight decay	0.3
warmup epoch	20
warmup learning rate	1.00E-05
label smoothing	0.1
mixup	0.8
cutmix	1
drop path	0.1
rand aug	9 0.5

Table 5. Hyperparameter Settings for DaViT-B model

B.3. Object Detection and Instance Segmentation

Hardware and Implementation: The experiments for Sec. 5.2 of the paper on object detection and instance segmentation were carried out on Nvidia A100 GPUs. The implementation followed [3]. We used the code base given by the GitHub [4] following their exact training regime.

Memory and Overheads: For this experiment we used a pre-trained XCiT-S and XCiT-M and thus the only overhead is given in the pre-training of the XCiT-S and XCiT-M architectures on ImageNet-1k. This is given in Appendix B.2 and is less than $0.5GB$.

B.4. Language Models

Hardware and Implementation: The language modeling experiments in Sec. 5.3 were all carried out on a Nvidia A6000 GPU. The Crammed-Bert was implemented following the original paper [2] and the original GitHub [1]. The training regime follows [1] using an AdamW optimizer and training for a total of 500000 iterations. The GPT-2 was implemented following [5] and was trained with an AdamW optimizer for 10000 iterations.

Memory and Overheads: For both language models the memory incurred by adding a correction matrix given in Appendix B.1 was less than 0.5 GB.

B.5. Nyströmformer

Hardware and Implementation: All the experiments were carried out on Nvidia A100 GPUs following the implementation and hyperparameter settings given in [7].

Memory and Overheads: For the Nyströmformer on each task on the LRA benchmark the overall addition in memory was less than 1 GB.

References

[1] Jonas Geiping. Cramming. `https://github.com/JonasGeiping/cramming`, 2023. 7

[2] Jonas Geiping and Tom Goldstein. Cramming: Training a language model on a single gpu in one day. In *International Conference on Machine Learning*, pages 11117–11143. PMLR, 2023. 7

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 7

[4] Matterport. Mask r-cnn implementation, 2017. 7

[5] Praveen Raja. Tiny-stories-gpt. `https://github.com/PraveenRaja42/Tiny-Stories-GPT`. 7

[6] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 5

[7] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Fei Tan, Glenn Fung, Vikas Singh, Xiaodong Yuan, Sungsoo Ahn Wang, Dimitris Papailiopoulos, and Katerina Fragkiadaki. Github repository, 2021. 7