L2 NORM GUIDED ADAPTIVE COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Although the human brain can adjust the amount of time and energy it uses to solve problems of varying complexity, many standard neural networks require a fixed computation budget regardless of the problem's complexity. This work introduces L2 Adaptive Computation (LAC), a new algorithm that adjusts the computation budget, by tracking changes in the L2 norm of a neural network's hidden state as layers are applied to the input. Unlike previous methods, LAC does not require additional trainable modules or auxiliary loss terms to make halting decisions. LAC matches the results of best-performing methods on a complex synthetic task and improves image classification accuracy while also increasing efficiency.

1 INTRODUCTION

The human brain relies on adaptivity in computation, enabling it to adjust the amount of time and energy dedicated to a problem based on its complexity. Conversely, standard neural networks tend to allocate the same computational budget to an input regardless of its complexity. There have been studies aimed to train models equipped with a halting mechanism that controls the computational steps based on the input's complexity and adapts the loss function to account for computational budget (Graves, 2016; Dehghani et al., 2018; Banino et al., 2021; Schuster et al., 2022; Xue et al., 2023). Following the same goal, in this work, we present L2 Adaptive Computation (LAC), a simple yet very effective method that enables dynamic number of computation steps. Unlike most prior work, LAC does not require adding any extra module with learnable parameters to make the halting decision and requires no change on the loss function. Instead, LAC tracks changes in the L2-norm (Euclidean norm) of the model's activations as a proxy for computational progress, enabling it to determine when to halt computations. LAC is developed in Scenic(Dehghani et al., 2022) and uses JAX(Bradbury et al., 2018) and Flax(Heek et al., 2020) and the code is available at `<url is removed for blind review>`.

2 LAC: L2 ADAPTIVE COMPUTATION

LAC requires a step function to perform, a step function can be any part of the model, e.g., a layer of the neural network. Consider S as a step function where $S(x, h_n) = h_{n+1}$ where h_0 is initial state. We set a maximum number of computational steps N that the model can take if it does not halt before reaching step N. To make the decision whether to continue to step n+1 or halt, LAC tracks the changes in the L2 norm of the model's activations($\|h_n\|_2$) up to step n.

L2 norm of model's activations LAC offers two different ways for making the halting decision, one is per-batch halting and the other is per-example (or optionally per-tokens), where depending on the hardware and framework, either could be preferred to efficiently introduce adaptivity. With per-batch halting, the L2-norm of the activation is calculated for all examples in the batch: $\|h\|_2 = \sqrt{\Sigma_{i=1}^{n}\Sigma_{j=1}^{d} h_{ij}^2}$, where $\|h\|_2 \in \mathbb{R}^1$, where $h \in \mathbb{R}^{n\times d}$, with d and n denoting the hidden dimension and the batch size respectively. With per-example halting, the L2-norm of the activations is calculated separately for each individual example: $\|h\|_2 = \{\sqrt{\Sigma_{j=1}^{d} h_{1j}^2}, ..., \sqrt{\Sigma_{j=1}^{d} h_{nj}^2}\}$, where $\|h\|_2 \in \mathbb{R}^n$.

Halting Mechanism In LAC, the decision to halt at each step is based on the model's *progress* during that step. Specifically, we use the change in the L2-norm of the model's activation as a proxy for progress, i.e., $\delta_n = \|h_n\|_2 - \|h_{n-1}\|_2$. In other words, we stop at step n if $\delta_n < \lambda_n$, where λ_n is a threshold defining the minimum activation change required for the model to be eligible to proceed

Table 1: Parity task.

Method	Accuracy	Steps
RNN	1.0	10
RNN + LAC per-batch	1.0	3
RNN + LAC per-example	1.0	3

Table 2: Image classification task.

Method	Accuracy
ViT	0.745
UViT	0.739
ViT + LAC per-batch	0.761
ViT + LAC per-example	0.759
UViT + LAC per-batch	0.743
UViT + LAC per-example	0.753

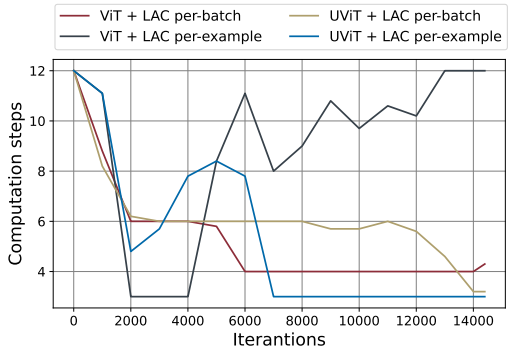

Figure 1: Number of steps taken by the model on CIFAR10 evaluation set with LAC per-example and LAC per-batch applied on UViT and Vit.

to step $n+1$. We set $\lambda_n = \alpha|\max(\Delta_n) - \min(\Delta_n)|$, where $\alpha \in (0, 1]$ is a hyperparameter that determines the decisiveness of the threshold, and Δ_n is the sequence of changes in activation across all consecutive steps before step n: $\Delta_n = \{\delta_{n-1}, \delta_{n-2} \ldots, \delta_1\}$. The value of α provides a knob to control the number of computational steps by adjusting the threshold. A lower value of α corresponds to a more permissive threshold, which in turn favors a larger number of computational steps.

3 Experiments and Results

This section provides a condensed report of the evaluation of LAC on two distinct tasks, Parity and image classification. In these experiments, LAC was implemented to introduce adaptability to various architectures such as RNN, Vision Transformer, and Universal Transformer.

Parity In this section, we present the results of applying LAC to the parity task, in which, given a sequence of numbers $\in \{1or-1, or0\}$, the model has to predict the evenness or oddness of the number of 1s in the sequence (Graves, 2016). The results of the parity task on three different architectures are presented in Table 1. RNN, RNN + LAC per-example, and RNN + LAC per-batch, where in the last two, we use a modified GRU cell to use LAC. All architectures have the same batch size of 128 and hidden state size of 128 and were tested on parity task with the sequence length of 16. The maximum number of computational steps, N, was set to 10.

Image Classification We further applied LAC to Vision Transformer(ViT) Dosovitskiy et al. (2020) as well as Universal Vision Transformer(UViT) (Dehghani et al., 2018) (which ties the parameters of the model across layers) to evaluate LAC in the context of the image classification task. We use the CIFAR10 dataset, and apply no data augmentations for simplicity of comparisons. We set our ViT/UViT configuration to standard B/16 where the model has 12 layers. Table 2 shows the evaluation accuracy of LAC on CIFAR10, compared to the baseline. LAC achieved similar accuracy compared to UViT and ViT while increasing efficiency via taking less number of computational steps (i.e., fewer layers are applied on average). Figure 1 presents the number of computational steps, in this case, encoder layers required for encoding images from the CIFAR dataset. Our approach exhibits a lower computational budget compared to UViT/ViT on the given inputs while maintaining the same level of performance.

4 Conclusion

Our paper introduces L2 Adaptive Computation (LAC), a new method that allows neural networks to adapt their computational budget based per patch, example, or token level. LAC utilizes the L2-norm of the model's activations for making halting decisions, eliminating the need for extra learnable parameters or auxiliary loss terms. Our experiments on two tasks, namely Parity and image classification, indicate that LAC achieves comparable results to the baselines while also improving efficiency. As an immediate future work, given the excellent cost-performance trade-off that LAC offers, we are seeking ways to access computational powers to apply LAC on large-scale training in the transfer learning setup.

REFERENCES

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Mostafa Dehghani, Alexey Gritsenko, Anurag Arnab, Matthias Minderer, and Yi Tay. Scenic: A jax library for computer vision research and beyond. 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL `http://github.com/google/flax`.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *arXiv preprint arXiv:2207.07061*, 2022.

Fuzhao Xue, Valerii Likhosherstov, Anurag Arnab, Neil Houlsby, Mostafa Dehghani, and Yang You. Adaptive computation with elastic input sequence. *arXiv preprint arXiv:2301.13195*, 2023.