Trainable Highly-expressive Activation Functions Supplementary Material

Irit Chelly*, Shahaf E. Finder*, Shira Ifergane, and Oren Freifeld

The Department of Computer Science, Ben-Gurion University of the Negev, Israel
{tohamy,finders,shiraif}@post.bgu.ac.il, orenfr@cs.bgu.ac.il

Abstract. This document contains the supplementary material for the main paper. In § A, we provide additional details about the regression and classification tasks in the toy-data experiments, such as data generation, training process scheme, result trends, function visualizations and further experiments. In § B, we provide additional details about the training procedures of the semantic segmentation and image generation tasks in the real-world data experiments. In § C we present various DiTAC versions, discuss their quality, and display their illustrations. In § D we measure DiTAC's computational cost on various configurations.

A Toy-data Experiments

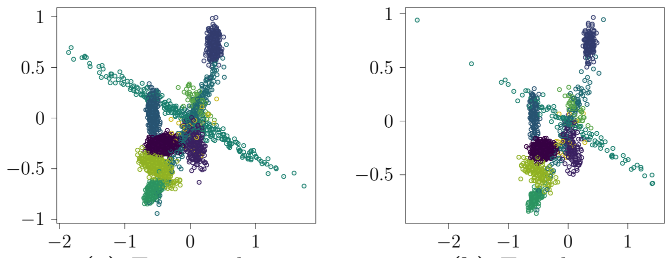
A.1 Classification

Here we provide additional details about the toy-data classification task presented in section 4.1 in the paper. We train a simple MLP on MNIST dataset and a synthetic dataset generated from a two-dimensional Gaussian-Mixture-Model (2D-GMM).

Data generation of 2D-GMM. We generate a synthetic data sampled from a 2D GMM with 10 groups (referred to as classes). The parameters of each Gaussian component (the mean vector and covariance matrix) are drawn from a normal-inverse-Wishart distribution while the mixture weights are drawn drawn from a Dirichlet distribution. We then draw $5 \cdot 10^3$ *i.i.d.* 2D points from the GMM, with a split of 70%/30% for train/test data. An example for such sample is shown in Fig. 8.

Training procedure. We use a simple MLP with two hidden layers to train on either MNIST or the 2D-GMM datasets. For MNIST we set their sizes to 128 and 64, and for the 2D-GMM we set both to 100. We train both datasets for 150 epochs, and use Adam optimizer, a batch size of 64, and a learning rate of $1 \cdot 10^4$ with no weight decay. As for the competitors, we use the same training parameters except for the learning rate: we run each configuration with several learning rates and select the best performance for each competitor.

*Equal contribution.



(a) Training data **(b)** Test data

Fig. 8: Here we display a 2D-GMM dataset sample separated into train and test sets.

Top-1 accuracy trend. In Fig. 9 we show top-1 accuracy results of DiTAC vs. existing competitors, trained on MNIST dataset. The results demonstrate a clear advantage to DiTAC.

A.2 Regression

Here we provide additional details about the toy-data regression task presented in section 4.1 in the paper, and display two more experiments that are not shown in the main text.

Auto-MPG dataset. We run a standard linear regression problem on a simple MLP with two hidden layers of size 100, on the Auto-MPG dataset [8], using the MPG and Horsepower values. Similar to other experiments, we compared DiTAC to existing AFs and TAFs. In Table 7 we report the Mean Squared Error (MSE) and the R-squared (R2) score, and show that DiTAC provides the best performance. Training details are shared later in this section. This experiment is not presented in the main text.

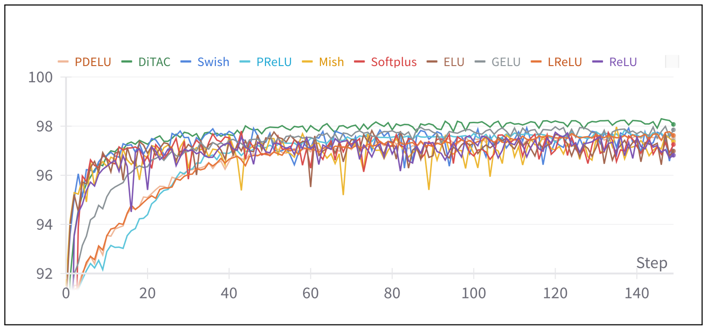


Fig. 9: Top-1 accuracy of MNIST classification on a simple MLP, using various AFs.

Table 7: Regression results on Auto-MPG dataset (horsepower) using a simple MLP. We were unable to report accuracy in several cases due to an unstable training process.

Act.	MSE ↓	R2 ↑
ReLU	409.0	71.8
LReLU	404.3	72.2
GELU	412.2	71.6
ELU	404.3	72.2
Softplus	410.0	71.8
Mish	410.9	71.7
Swish	405.8	72.1
PReLU	406.3	72.0
PDELU	–	–
DiTAC	**389.6**	**73.2**

Reconstruction of 1D/2D functions. We experiment with regression tasks of reconstructing one- and two-dimensional functions. Here we elaborate more details about the experiments mentioned in the paper (Table 2 in section 4.1), and show an experiment on a new 1D function. Training details are shared later in this section.

In Table 2 in the paper, we display performance evaluation of DiTAC vs. other AFs and TAFs in reconstructing 1D and 2D target functions, by training an MLP with one hidden layer of sizes 30 and 50 respectively. The 1D target function is $\sin(\exp 6x)$, and the 2D target function is a sum of sines with various frequencies:

$$0.4\sin(9xy) + 0.1\sin(-9x + 11y) + 0.15\sin(3x + 13y) \tag{1}$$
$$+ 0.15\sin(9x + 9y) + 0.1\sin(13x + 5y) + 0.1\sin(3x + 19y) \tag{2}$$

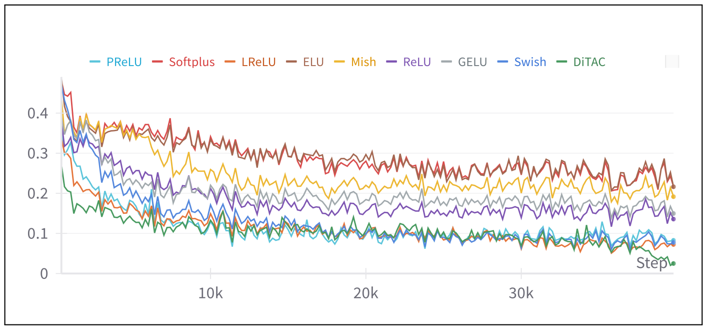
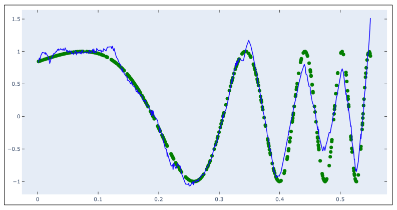


Fig. 10: MSE of the 1D-function reconstruction task, $\sin(\exp 6x)$, on a simple MLP, using various AFs.


(a) DiTAC

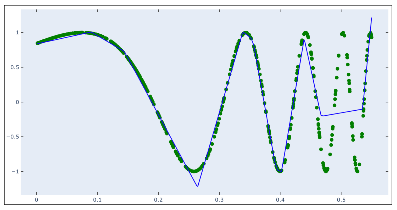

(b) LReLU

Fig. 11: 1D-function reconstruction, $\sin(\exp 6x)$, learned by DiTAC and (the runner-up) LReLU. DiTAC manages to learn a smooth function that fits the data.

In Fig. 10 we present the MSE trend of DiTAC vs. existing AFs and TAFs, on the 1D-function reconstruction experiment. As shown in Table 2 in the paper, DiTAC noticeably provides the best performance, gaining R2 of 96.26% vs. 89.78% by LReLU (second best) in the 1D-function experiment, and R2 of 98.37% vs. 96.09% by PReLU (second best) in the 2D-function experiment. This performance gap is also demonstrated in Fig. 11, where we compare the reconstructed 1D-function learned by DiTAC and its runner-up LReLU.

We show another 1D-function reconstruction experiment, where the target function is as follows:

$$0.4\sin(19x) + 0.2\sin(23x) + 0.3\sin(29x) + 0.1\sin(31x) \tag{3}$$

Here we used an MLP with one hidden layer of size 64. Results and model size (number of parameters) are shown in Table 8 and Fig. 12. It can be shown that only part of the competitors manage to fit a reasonable function, while the others perform badly, although each AF was tested on several learning rate options.

Table 8: Regression-task results of learning one-dimensional target function on a simple MLP, using various AFs and TAFs, along with the number of model parameters used by each activation.

AF/TAF	Param.	MSE ↓	R2 ↑
ReLU	193	0.010	93.3
LReLU	193	0.008	94.7
GELU	193	0.111	9.7
ELU	193	0.116	3.6
Softplus	193	0.114	5.2
Mish	193	0.113	5.5
Swish	194	0.004	97.5
PReLU	194	0.010	93.1
DiTAC	202	**0.001**	**99.3**

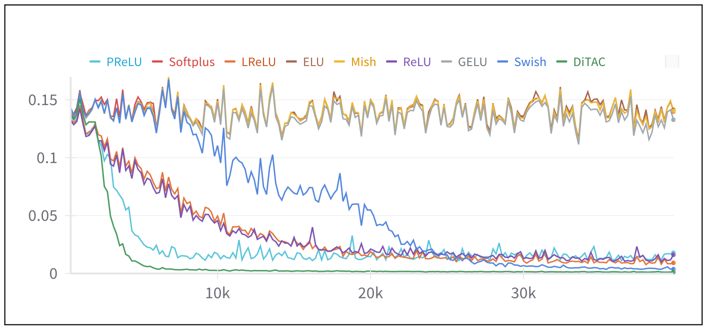


Fig. 12: MSE of the 1D-function reconstruction task on a simple MLP, using various AFs.

Training procedure. In all of the aforementioned regression experiments, we train a model for 40K iterations, using Adam optimizer, a batch size of 98, no weight decay, and a learning rate of 0.01 except of the new 1D-function experiment, in which we use a learning rate of $1 \cdot 10^3$. Similar to the classification experiments, we evaluate the competitors using the same training parameters except for the learning rate, as each competitor's best performance was selected out of several learning rates options.

B Real-world Data Experiments

Here we provide technical details about the training procedures of the semantic segmentation and the image generation tasks.

Semantic Segmentation. In the paper we compare the performance of DiTAC, ReLU and GELU activation functions on the semantic segmentation task. In the two first experiments we train UperNet [10] as a segmentation framework on the ADE20K dataset [12], where in one configuration we use ConvNeXT-T and in the other we use Swin-T as the backbone. We follow `mmseg`'s [3] configuration to setup the training parameters. We use 160K iterations schedule, crop size of 512×512, batch size of 16, weight decay of 0.01, and the polynomial learning rate decay policy [2] where the initial learning rate is $6 \cdot 10^{-5}$ and the power is 1. In the third experiment we train a PSPNet [11] segmentation framework using ResNet-50 [7] as the backbone, on the Cityscapes dataset [4]. We use a 40K-iteration schedule, crop size of 512×1024, batch size of 8, weight decay of $5 \cdot 10^{-4}$, and the polynomial learning rate decay policy where the initial learning rate is 0.01 and the power is 0.9. In all configurations the backbones are pre-trained on ImageNet-1K dataset using 300 epoch schedule.

Generative Networks. In the paper we evaluate DiTAC on the image generation task, on DCGAN [9], an unconditional GAN, and on BigGAN [1], a conditional GAN. We train DCGAN for 300K iterations on 4 GPUs, using image crop size of 64×64, an effective batch size of 128, Adam optimizer, and a learning rate of $2 \cdot 10^{-4}$ in both D and G. We train BigGAN for 500K iterations, using an effective batch size of 64, Adam optimizer, and a learning rate of $2 \cdot 10^{-4}$ in D and $5 \cdot 10^{-5}$ in G.

C DiTAC Versions

In the paper, in section 3.2, we present how DiTAC is built. DiTAC uses a CPAB transformation $T^{\boldsymbol{\theta}}$, which is defined on a finite interval, $\Omega = [a, b] \subset \mathbb{R}$, and its co-domain is also a finite interval. In order to handle input data that fall outside of $[a, b]$, we combine $T^{\boldsymbol{\theta}}$ with GELU, a recent widely-used AF in state-of-the-art models. Other versions of DiTAC can also be built by combining $T^{\boldsymbol{\theta}}$ with a variety of other AFs, or defining a certain function, not necessarily a known AF, outside of CPAB's map. Here we present additional DiTAC versions (definitions and illustrations), share our insights about their quality, and evaluate their performance on a simple regression task. All DiTAC versions are illustrated in Fig. 13.

We note, that conceptually it is possible to apply CPAB transformation on the entire input data by first apply a normalization that maps the data into Ω, perform the CPAB transformation, and then rescale the transformed data back to its original range. However, we will then have to extract the minimum and maximum values of the entire input data, something that is hard to achieve during training as these values depend on the network's parameters learning. Therefore, we set the $[a, b]$-interval before training, which usually includes a large portion of the data, and apply a different behavior on the data that is outside of that range.

GELU-like DiTAC (DiTAC). This is the main DiTAC version, which is also used in all of our experiments. GELU is a natural choice given its popularity and success in state-of-the-art architectures. Input data that fall outside of the $[a, b]$-interval inherit GELU's behavior, while the input data inside $[a, b]$ first go through CPAB transformation and then go through the GELU function. The GELU-like DiTAC is defined as follows:

$$\text{DiTAC}(x) = \tilde{x} \cdot \Phi(x) \,, \qquad \tilde{x} = \begin{cases} T^{\boldsymbol{\theta}}(x) & \text{If } a \leq x \leq b \\ x & \text{Otherwise} \end{cases} \tag{4}$$

where Φ is the cumulative distribution function (CDF) of a standard normal distribution, $T^{\boldsymbol{\theta}}$ is a CPAB transformation and $\Omega = [a, b]$, the domain of $T^{\boldsymbol{\theta}}$, is user-defined.

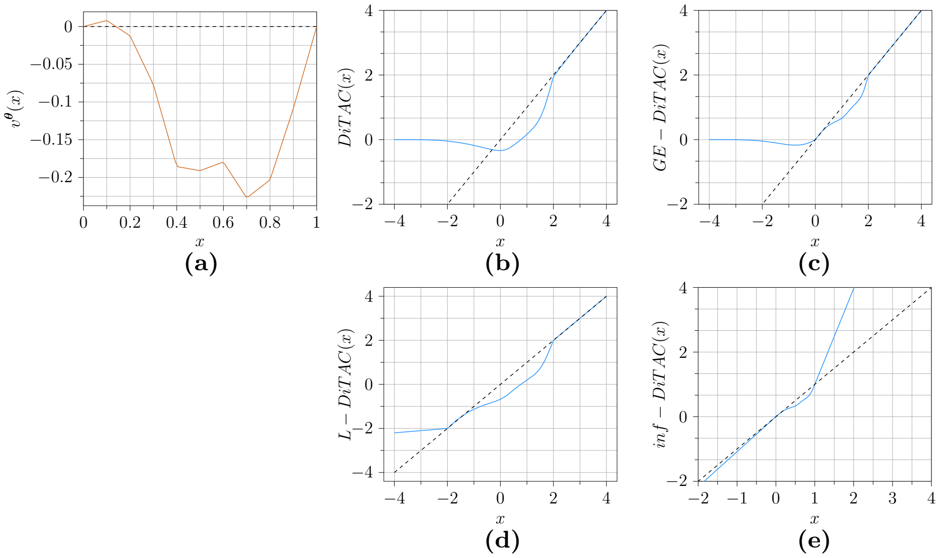


Fig. 13: Illustration of DiTAC versions. We display (a) $v^{\boldsymbol{\theta}}$, the CPA velocity field used in all DiTAC version illustrations, (b) DiTAC, (c) GE-DiTAC, (d) L-DiTAC, and (e) inf-DiTAC

GELU-DiTAC (GE-DiTAC). This activation is similar to DiTAC's main version, except that here we apply GELU only on negative input values, whereas a pure CPAB transformation is performed on the input-data range $[0, b]$. In order to keep the function continuous (in case we impose zero-boundary conditions on $\boldsymbol{v}^{\boldsymbol{\theta}}$; see [6]), we apply the identity function on values that are larger than b. The GE-DiTAC is defined as follows:

$$\mathrm{GE-DiTAC}(x) = \begin{cases} x \cdot \varPhi(x) & \text{If } x < 0 \\ T^{\boldsymbol{\theta}}(x) & \text{If } 0 \leq x \leq b \\ x & \text{If } x > b \end{cases} \tag{5}$$

where $\varPhi$ is the CDF of a standard normal distribution, $T^{\boldsymbol{\theta}}$ is a CPAB transformation and $\varOmega = [0, b]$, the domain of $T^{\boldsymbol{\theta}}$, is user-defined.
Note that GE-DiTAC allows CPAB transformation's capability to be more evident, as this part of the transformed data is not composed with any other function. Empirically, it performs similar to DiTAC in most of our experiments, while its advantage is mainly demonstrated in simple regression tasks using simple networks.

Leaky DiTAC (L-DiTAC). Here $T^{\boldsymbol{\theta}}$ is applied on $[a, b]$ while the rest of the data goes through a Leaky-ReLU (LReLU) function. That is,

$$\text{Leaky DiTAC}(x) = \begin{cases} T^{\boldsymbol{\theta}}(x) & \text{If } a \leq x \leq b \\ \text{LReLU}(x) & \text{Otherwise} \end{cases} \tag{6}$$

where $T^{\boldsymbol{\theta}}$ is a CPAB transformation and $\Omega[a, b]$, the domain of $T^{\boldsymbol{\theta}}$, is user-defined. This version can be as an expressive version of ReLU. As shown in [5], different AFs are suitable to different types of data and tasks. This DiTAC version may improve problems in which the ReLU function performs better than any other existing AF.

Infinite-edges DiTAC (inf-DiTAC). Recall that CPAB transformations are obtained via an integration of elements in $\mathcal{V}$, which is a space of continuous functions from Ω to $\mathbb{R}$, that are piecewise-affine w.r.t. some fixed partition of Ω into sub-intervals. In inf-DiTAC, similarly to GE-DiTAC and L-DiTAC, $T^{\boldsymbol{\theta}}$ is applied on $[a, b]$-interval. As for the input data that fall outside of that range, we apply the affine transformations learned in both edges of Ω's tessellation (most-right and most-left cells), yielding a continuous AF which is controlled solely by CPAB transformation parameters. The inf-DiTAC is defined as follows:

$$\text{inf} - \text{DiTAC}(x) = \begin{cases} A_l^{\boldsymbol{\theta}} x & \text{If } x < a \\ T^{\boldsymbol{\theta}}(x) & \text{If } a \leq x \leq b \\ A_r^{\boldsymbol{\theta}} x & \text{If } x > b \end{cases} \tag{7}$$

where $T^{\boldsymbol{\theta}}$ is a CPAB transformation, $A_l^{\boldsymbol{\theta}}$ and $A_r^{\boldsymbol{\theta}}$ are the affine transformations in the most-left and most-right cells of the tessellation respectively, and $\Omega[a, b]$, the domain of $T^{\boldsymbol{\theta}}$, is user-defined.

Table 9: Regression-task results of learning a two-dimensional target function, using various DiTAC versions on a simple MLP.

DiTAC Version	MSE ↓	R2 ↑
DiTAC	0.004	98.4
GE-DiTAC	0.001	99.4
L-DiTAC	0.006	98.7
inf-DiTAC	0.004	97.6

In Table 9 we show performance evaluation of all aforementioned DiTAC versions, on the two-dimensional function-reconstruction task presented in the paper (training details are provided in § A.2). It can be shown that GE-DiTAC provides the best performance on this specific task.

D Computational Cost

In section 3.3 in the paper we extensively describe DiTAC's computational cost, and the use of a lookup table in both training and inference phases. In Table 10 we measure the computational cost by comparing the # of parameters, FLOPs and latency during inference, along with top-1 accuracy on ImageNet-1K, where DiTAC is integrated in two model types: mobile-oriented (*e.g.* MobileNet-V3), and massive architectures (*e.g.* ConvNeXT-S). Evidently, DiTAC consistently improves accuracy with a moderate increase in latency, and no significant change in FLOPs or Params.

Table 10: Computational cost/accuracy (on NVIDIA Tesla P100 GPU, batch size of 1). Models using DiTAC are marked with $^+$.

Configuration	Params.	FLOPs	Latency [sec]	Top-1
MobileNet-V3	1.57M	57.2M	0.009±0.004	61.1
MobileNet-V3$^+$	1.57M	60.4M	0.016±0.001	**61.3**
ConvNeXT-S	49.5M	8.6B	0.016±0.002	83.1
ConvNeXT-S$^+$	49.5M	8.7B	0.027± 0.001	**83.3**

References

1. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096 (2018)
2. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE TPAMI (2017)
3. Contributors, M.: MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation (2020)
4. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: CVPR (2016)
5. Dubey, S.R., Singh, S.K., Chaudhuri, B.B.: Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing (2022)
6. Freifeld, O., Hauberg, S., Batmanghelich, K., Fisher III, J.W.: Transformations based on continuous piecewise-affine velocity fields. IEEE TPAMI (2017)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
8. Quinlan, R.: Auto MPG. UCI Machine Learning Repository (1993), DOI: https://doi.org/10.24432/C5859H
9. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)

10. Xiao, T., Liu, Y., Zhou, B., Jiang, Y., Sun, J.: Unified perceptual parsing for scene understanding. In: ECCV (2018)
11. Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: CVPR (2017)
12. Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., Torralba, A.: Semantic understanding of scenes through the ade20k dataset. IJCV (2019)