Differentiable Programming for Piecewise Polynomial Functions

Minsu Cho[*1] **Ameya Joshi**[*1] **Xian Yeow Lee** [2] **Aditya Balu** [2] **Adarsh Krishnamurthy** [2]

Baskar Ganapathysubramanian [2] **Soumik Sarkar** [2] **Chinmay Hegde** [1]

[1] New York University [2]Iowa State University

Abstract

We introduce a new, principled approach to extend gradient-based optimization to piecewise smooth models, such as k-histograms, splines, and segmentation maps. We derive an accurate form of the weak Jacobian of such functions and show that it exhibits a block-sparse structure that can be computed implicitly and efficiently. We show that using the redesigned Jacobian leads to improved performance in applications such as denoising with piecewise polynomial regression models, data-free generative model training, and image segmentation.

1 Introduction

Differentiable programming has been proposed as a paradigm shift in algorithm design. The main idea is to leverage gradient-based optimization to optimize the parameters of the algorithm, allowing for end-to-end trainable systems (such as deep neural networks) to exploit structure and patterns in data and achieve better performance [13, 12, 19, 17, 8, 3, 16, 6]. One way to leverage differential programming modules is to encode additional structural priors as "layers" in a larger machine learning model.

In these contexts, however, the space of possible priors that can be encoded are restricted to be differentiable (by definition), and this poses a major limitation. For example, consider applications such as calculation of summary statistics in data streams. A popular prior for such statistics is that they are well-approximated by piecewise constant, or piecewise- linear functions, for which there exist fast and optimal algorithms [15].

To fully leverage differentiable programming for such problems, we would like to compute gradients "through" these approximation algorithms. Generally, embedding such discontinuous co-domains functions as layers in a differentiable program (such as backpropagation) requires special care. A popular solution is to relax these non-differentiable, discrete components into continuous approximations for whom gradients exist [4, 7, 18, 10, 9, 2]. However, for discrete, non-differentiable objectives such as those encountered in piecewise regression, these approaches are not applicable.

We propose a principled approach for differentiable programming with piecewise polynomial priors. For the forward pass, we leverage *fast* algorithms for computing the optimal projection of any given input onto the space of piecewise polynomial functions. For the backward pass, we observe that every piece (partition) in the output approximation only involves input elements from the same piece. Using this, we derive a *weak* form of the Jacobian for the piecewise polynomial approximation operator. While we focus on piecewise polynomial approximation in this paper, our approach can be generalized to any algorithmic module with piecewise outputs with differentiable subfunctions.

[*]Equal Contribution. Emails: {mc8065, ameya.joshi}@nyu.edu

1st Workshop on Learning Meets Combinatorial Algorithms @ NeurIPS 2020, Vancouver, Canada.

2 Differentiable Piecewise Polynomial Approximation

We introduce Differentiable Piecewise Approximation (DPA) as an approach to estimate gradients over piecewise polynomial function approximators. Our main goal will be to estimate an analytical form of the (weak) Jacobian of piecewise polynomial approximation, enabling us to use such function approximators within backward passes in general differentiable programs.

First, we set up some preliminaries.

Def. 1 (Piecewise polynomials)**.** A function $h : \mathbb{R}^n \to \mathbb{R}^n$ is said to be k-piecewise polynomial with degree d if $h(\cdot)$ is the sum of sub-functions $h = \sum_{i=1}^{k} h_{\mathbf{B}_i}(\cdot)$, where $\Pi_{\mathbf{B}} = \{\mathbf{B}_1, \ldots, \mathbf{B}_k\}$ is a partition of $[n]$, $h_{\mathbf{B}_i}$ is a degree d polynomial if there exists k corresponding parameters $\alpha_1, \ldots, \alpha_k \in \mathbb{R}^n$, and a fixed, known j such that for all index vector $\mathbf{t} = (t_1, t_2, \ldots, t_n) \in \mathbb{R}^n$ and associated vector $\mathbf{v}_j = (1, t_j, t_j^2, \ldots, t_j^d)$, $[h_{\mathbf{B}_i}]_j = \langle \alpha_i, \mathbf{v}_j \rangle$ if $t_j \in \mathbf{B}_i$. We denote $\mathcal{F}_d$ as the family of all such piecewise polynomial functions h.

Forward pass: A key component of our approach is the *orthogonal projection* onto the nonlinear manifold (specifically, the union-of-subspaces) defined by $\mathcal{F}_d^k$. If $h(\cdot)$ is the best-fit k-polynomial approximation for $\mathbf{x} \in \mathbb{R}^n$, we can find $h(\cdot)$ (as parameterized by B_i) by solving the optimization problem:

$$\hat{h}(\mathbf{x}) = \arg\min_{h} \frac{1}{2} \|\mathbf{x} - h\|_2^2 \quad \text{s.t. } h \in \mathcal{F}_d^k$$

Despite the fact that this is a non-convex operation, such an orthogonal projection can be computed in polynomial time using a few different techniques (including dynamic programming; we describe details in specific cases below). This forms the *forward pass* of our DPA "layer".

Backward pass: We now turn to the backward pass. To calculate the Jacobian of $h(\mathbf{x})$ with respect to $\mathbf{x}$, there are two properties of such projection functions that we leverage: (1) $h(\cdot)$ induces a partition of $\mathbf{x}$, that is, each element of $\mathbf{x}$ corresponds to a *single* piece, $h_{\mathbf{B}_i}(\mathbf{x})$, and (2) the sub-function in each partition is continuous and differentiable.

The first property ensures that that every element $\mathbf{x}_i$ contributes to only a single piece in the output $h(\mathbf{x})$. Given that the sub-functions from piecewise partitioning function are smooth, we also observe that the size of each block corresponds to the size of the partition, $\mathbf{B}_i$. Using this observation, we get the following statement (see Appendix for proofs).

Theorem 1 (Jacobian for DPA)**.** *For any piecewise polynomial function, $h : \mathbb{R}^n \to \mathbb{R}^n$, that takes as input a vector $\mathbf{x}$, and outputs a piecewise approximation, $h(\mathbf{x})$ with k partitions, the Jacobian can be expressed as a* block diagonal *matrix, $\mathbf{J} \in \mathbb{R}^{n \times n}$ such that,*

$$\mathbf{J}_{\mathbf{x}}(h(\mathbf{x}))_{s,t} = \frac{\partial h(\mathbf{x})_s}{\partial \mathrm{x}_t} = \begin{cases} \frac{\partial h_{\mathbf{B}_i}(\mathbf{x})_s}{\partial x_t} & \text{if } h(\mathbf{x})_s, \mathrm{x}_t \in \mathbf{B}_i \\ 0 & \text{if } h(\mathbf{x})_s, \mathrm{x}_t \notin \mathbf{B}_i \end{cases} \tag{1}$$

1D piecewise constant regression: First, we consider the case of piecewise regression in 1D, where we can use any algorithm to approximate a given input vector with a fixed number of piecewise polynomial functions. The simplest example is that of piecewise *constant* regression, where a given input vector is approximated by a set of constant segments. A *k-histogram approximation algorithm* searches over the collection of all partitions $\Pi = \{\mathbf{B}_1, \ldots \mathbf{B}_k\}$ of $[n]$ such that $\sum_{i=1}^{k} \|h_{\mathbf{B}_i}(\mathbf{x}) - \mathbf{x}_{\mathbf{B}_i}\|$ is minimized where $[\mathbf{x}_{\mathbf{B}_i}]_j = [\mathbf{x}]_j \mathbb{1}(j \in \mathbf{B}_i)$. Equivalently, the algorithm solves the following optimization problem:

$$\min_{\mathbf{B}_1, \ldots, \mathbf{B}_k} \sum_{i=1}^{k} \sum_{j \in \mathbf{B}_i} (x_j - \frac{1}{|\mathbf{B}_i|} \sum_{l \in \mathbf{B}_i} x_l)^2$$

We assume an optimal $h(\cdot)$ (parameterized by $\{\mathbf{B}_i\}$) that can be obtained using many existing methods (a classical approach is by dynamic programming [14]). The running time of such approaches is typically $O(nk)$, which is constant for fixed k; see [1] for a more detailed treatment.

Using Theorem 1, the Jacobian of the output k-histogram with respect to $\mathbf{x}$ assumes the following form:

$$\frac{\partial h(\mathbf{x})}{\partial x_j} = \frac{\partial}{\partial x_j}\sum_{i=1}^{k}\left(\frac{1}{|\mathbf{B}_i|}\sum_{l\in\mathbf{B}_i} x_l \mathbb{1}_{\mathbf{B}_i}\right) = \sum_{i=1}^{k}\frac{\partial}{\partial x_j}\frac{1}{|\mathbf{B}_i|}\sum_{l\in B_i} x_l \mathbb{1}_{\mathbf{B}_i} = \frac{1}{|\mathbf{B}_i|}\mathbb{1}_{\mathbf{B}_i}$$

where $\mathbb{1}_{\mathbf{B}_i}$ is a vector populated with ones with the index corresponding to $\mathbf{B}_i$ and zeros otherwise. Therefore, the Jacobian of h with respect to $\mathbf{x}$ forms the block-diagonal matrix $\mathbf{J} \in \mathbb{R}^{n\times n}$:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{J}_k \end{bmatrix}$$

where all entries of $\mathbf{J}_i \in \mathbb{R}^{|\mathbf{B}_i|\times|\mathbf{B}_i|}$ equal to $1/|\mathbf{B}_i|$. Note here that the sparse structure of the Jacobian allows for fast computation, and it can be easily seen that computing the Jacobian vector product $\mathbf{J}^T\nu$ for any input ν requires $O(n)$ running time. As an additional benefit, the decoupling induced by the partition enables further speed up in computation via parallelization.

Generalization to 1D piecewise polyomial fitting We derive differentiable forms of generalized piecewise d-polynomial regression. Let $h : \mathbb{R}^n \to \mathbb{R}^n$ be any algorithm to compute the k-piecewise polynomial approximation of an input vector $\mathbf{x}$ that outputs partitions $\mathbf{B} = \{\mathbf{B}_1, \ldots, \mathbf{B}_k\}$. Then, for each partition, we are required to solve a d-degree polynomial regression. Generally, the polynomial regression problem is simplified to linear regression by converting input data into a Vandermonde matrix. Assume that for partition, $\mathbf{B}_i$, the input indices $\{t_1, t_2, \ldots, t_{|\mathbf{B}_i|}\}$ are represented as a Vandermonde matrix, $\mathbf{V}_{\mathbf{B}_i}$.

The optimal coefficients corresponding to partition $\mathbf{B}_i$ is $\alpha_{\mathbf{B}_i} = (\mathbf{V}_{\mathbf{B}_i}^T\mathbf{V}_{\mathbf{B}_i})^{-1}\mathbf{V}_{\mathbf{B}_i}^T\mathbf{x}_{\mathbf{B}_i}$ which can be computed in $O(knd^\omega)$ time where ω is the matrix-multiplication exponent [11].

Using Theorem 1 and the gradient for polynomial regression, the Jacobian of $h_{\mathbf{B}_i}(\mathbf{x})$ with respect to $\mathbf{x}$ forms a blockwise sparse matrix: (we defer the detailed derivation in the appendix)

$$\frac{\partial[h_{\mathbf{B}_i}]_j}{\partial x_l} = \begin{cases} \left[\mathbf{V}_{\mathbf{B}_i}(\mathbf{V}_{\mathbf{B}_i}^T\mathbf{V}_{\mathbf{B}_i})^{-1}[\mathbf{V}_{\mathbf{B}_i}^T]_j\right]_l & \text{if } \ell \in \mathbf{B}_i \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The two main takeaways here are as follows: (1) $\mathbf{V}_{B_i}$ can be precomputed for all possible $n-1$ partition sizes, thus allowing for fast ($O(n)$) computation of Jacobian-vector products; and (2) an added flexibility is that we can independently control the degree of the polynomial used in each of the partitions. The second advantage could be very useful for heterogeneous data as well as considering boundary cases in data streams.

3 2D piecewise constant functions

We leverage our approach to enforce deep models to predict piecewise constant segmentation maps. In the case of 2D images, note that we do not have a standard primitive (for piecewise constant fitting) to serve as the forward pass. Instead, we leverage connected-component algorithms (such as Hoshen-Kopelman, or other, techniques [21]) to produce a partition, and the predicted output is a piecewise constant image with values representing the mean of input pixels in the corresponding piece. For the backward pass, we use a tensor generalization of the block Jacobian where each partition is now represented as a channel which is only non-zero in the positions corresponding to the channel. Formally, if the image $\mathbf{x} \in \mathbb{R}^n$ is represented as the union of k partitions, $h(\mathbf{x}) = \bigcup_{i=1}^{k}\mathbf{B}_i$, the Jacobian, $\mathbf{J}(\mathbf{x}) = \partial h(\mathbf{x})/\partial\mathbf{x} \in \mathbb{R}^{n\times n}$ and,

$$(\mathbf{J})_{ij} = \begin{cases} 1/|\mathbf{B}_k| \text{ if } \ h(\mathbf{x})_i \in \mathbf{B}_k, \\ 0 \ \text{ otherwise.} \end{cases} \tag{3}$$

Note that $\mathbf{B}_i$ here no longer correspond to single blocks in the Jacobian. Here, they will reflect the positions of pixels associated with the various components. However, the Jacobian is still sparsely structured, enabling fast vector operations.

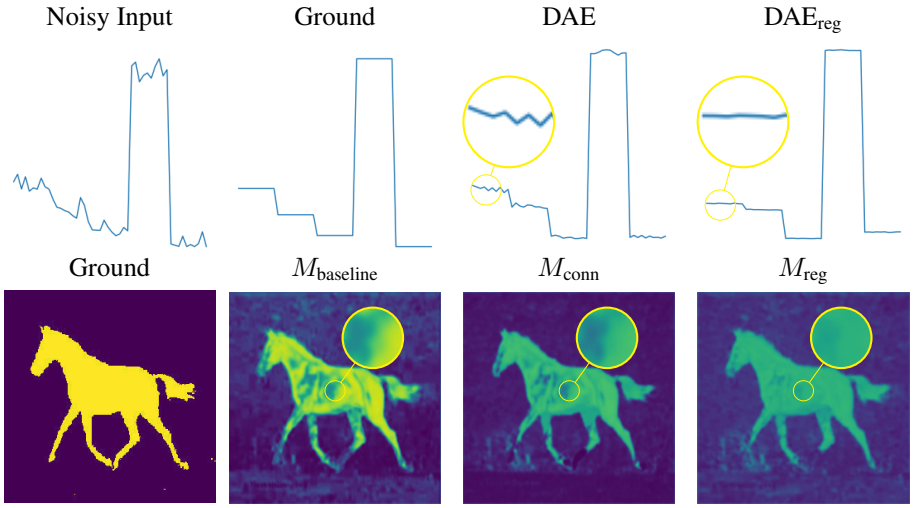


Figure 1: ***Denoising/Segmentation results*** *The first row corresponds to denoising of 1D piecewise constnat signals. We observe that training DAE with a regularizer with piecewise layer denoises the perturbed input signals better than vanilla DAE. The second row corresponds to segmentation results on Weizmann horse dataset. The two models, $M_{baseline}$ and M_{conn} were trained with and without the differentiable connected component layer respectively. M_{reg} minimizes the difference between the network output and its piecewise approximation along with the standard segmentation error. Note that M_{conn} and M_{reg} generate better segmentation masks with fewer holes. Also note the cleaner edges compared to the standard segmentation results. Additional figures are in the Appendix.*

Table 1: ***Quantitative segmentation performance:*** *Jaccard scores for the baseline and the connected component models. Predictions are thresholded at 0.25 (calculated on a validation set). Models trained with our DPA layer achieve higher scores.*

Model		M_{baseline}	M_{conn}	M_{Reg}
Jaccard scores	128×128	0.4986 ± 0.01	0.4810 ± 0.004	$\mathbf{0.4991 \pm 0.009}$
	256×256	$\mathbf{0.5056 \pm 0.006}$	0.4973 ± 0.008	0.4998 ± 0.006

4 Experiments

1D signal denoising: We train a *structured* denoising autoencoder (DAE) to denoise and reconstruct synthetic 1D signals, using our approach to enforce piecewise smooth structures. Let f be a DAE, h be a histogram layer, $\mathbf{x}$ and $\mathbf{y}$ be noisy input and clean signal respectively. We compare denoising performance in two settings: (1) vanilla DAE where the loss is defined as $\frac{1}{2}\|f(\mathbf{x})r - \mathbf{y}\|_2^2$ (DAE) and (2) a DAE along with a piecewise constant layer with additional MSE loss $\frac{1}{2}\|f(\mathbf{x}) - h(f(\mathbf{x}))\|_2^2$ (DAE$_{\text{reg}}$). We train the DAE with the synthetic dataset with piecewise constant/linear priors with additive Gaussian noise. Figure 1 compares the denoised inputs between DAE and DAE$_{\text{reg}}$. We observe that DAE$_{\text{reg}}$ further smooths the denoised input compare to the vanilla version.

Image segmentation: We use a similar setup as in [10] with a three-layer fully convolutional neural network; however, we use the mean-squared error for training instead of cross-entropy. We analyze the efficacy of our approach in two settings: (1) adding a piecewise constant approximation layer as the final layer of our network (M_{conn}), and (2) using the piecewise layer alongside the actual output as a regularizer (M_{reg}). We compare these with the baseline model without the piecewise constant layer.

We train the three models for two image sizes on the Weizmann horse dataset [5] using mean-squared error between the ground truth and the predicted segmentation map. For a fair comparison, we use the same base architecture and hyper-parameters for all models (see Appendix for details). Our piecewise approximation layer provides more consistent segmentation maps with fewer holes for both M_{conn} and M_{reg}. (see Figure 1).

References

[1] J. Acharya, I. Diakonikolas, C. Hegde, J. Z. Li, and L. Schmidt. Fast and near-optimal algorithms for approximating distributions by histograms. In *Proc. ACM SIGMOD-SIGACT-SIGAI Symp. on Principles of Database Systems*, 2015.

[2] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019.

[3] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[4] M. Blondel, O. Teboul, Q. Berthet, and J. Djolonga. Fast differentiable sorting and ranking. *ArXiv*, abs/2002.08871, 2020.

[5] E. Borenstein and S. Ullman. Learning to segment. In *Euro. Conf. Comp. Vision*, pages 315–328. Springer, 2004.

[6] W. Chen, J. Gao, H. Ling, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019.

[7] M. Cuturi, O. Teboul, and J.-P. Vert. Differentiable ranking and sorting using optimal transport. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019.

[8] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13, 2017.

[9] B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi. CvxNet: Learnable convex decomposition. In *IEEE Conf. Comp. Vision and Pattern Recog*. IEEE, 2020.

[10] J. Djolonga and A. Krause. Differentiable learning of submodular models. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, pages 1013–1023, 2017.

[11] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. on Database Systems (TODS)*, 31(1):396–438, 2006.

[12] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt. A differentiable programming system to bridge machine learning and scientific computing. *ArXiv*, abs/1907.07587, 2019.

[13] M. J. Innes. Algorithmic differentiation. In *Proc. Conf. ML. and Systems. (MLSys)*, 2020.

[14] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. of Int. Conference on Very Large Data Bases (VLDB)*, 1998.

[15] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proc. of IEEE Int. Conf. on data mining*, pages 289–296. IEEE, 2001.

[16] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen. Differentiable Monte-Carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6):1–11, 2018.

[17] T.-M. Li, M. Gharbi, A. Adams, F. Durand, and J. Ragan-Kelley. Differentiable programming for image processing and deep learning in halide. *ACM Trans. Graph.*, 37(4):1–13, 2018.

[18] A. Mensch and M. Blondel. Differentiable dynamic programming for structured prediction and attention. *ArXiv*, abs/1802.03676, 2018.

[19] F. Schafer, M. Kloc, C. Bruder, and N. Lorch. A differentiable programming method for quantum control. *ArXiv*, 2020.

[20] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 2014.

[21] K. Wu, E. Otoo, and A. Shoshani. Optimizing connected component labeling algorithms. In *Medical Imaging 2005: Image Processing*, volume 5747, pages 1965–1976. International Society for Optics and Photonics, 2005.

A Proofs and Derivations

Proof for Theorem 1

Proof. The proof follows similar arguments as in Proposition 4 from [4].

Let $\Pi_h = \{\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_k\}$ be k partitions induced by some $h : \mathbb{R}^n \to \mathbb{R}^n$ for some input, $\mathbf{x}_i$ and $h_{\mathbf{B}_j}$ be the sub-function associated with partition $\mathbf{B}_j$. Then, each element, x_i uniquely belongs to some partition $\mathbf{B}_s$.

Now,

$$\begin{aligned}\frac{\partial h(\mathbf{x})}{\partial \mathrm{x}_i} &= \frac{\partial \sum_{j=1}^{k} \mathbb{1}(\mathrm{x}_i \in \mathbf{B}_j) h_{\mathbf{B}_j}(\mathbf{x})}{\partial \mathrm{x}_i} \\ &= \begin{cases} \frac{\partial h_{\mathbf{B}_s}(\mathbf{x})}{\partial \mathrm{x}_i} & \text{if } \mathrm{x}_i \in \mathbf{B}_s \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

Note that this is a block-diagonal matrix with each block being $|\mathbf{B}_j| \times |\mathbf{B}_j|$, giving us the required statement.

□

Derivation for equation 2

Let $h : \mathbb{R}^n \to \mathbb{R}^n$ be any algorithm to compute the k-piecewise polynomial approximation of an input vector $\mathbf{x}$ that outputs partitions $\mathbf{B} = \{\mathbf{B}_1, \ldots, \mathbf{B}_k\}$. Assume that for partition, $\mathbf{B}_i$, the input indices $\{t_1, t_2, \ldots, t_{|\mathbf{B}_i|}\}$ are represented as a Vandermonde matrix, $\mathbf{V}_{\mathbf{B}_i}$.

$$\mathbf{V}_{\mathbf{B}_i} = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^d \\ 1 & t_2 & t_2^2 & \cdots & t_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_{|\mathbf{B}_i|} & t_{|\mathbf{B}_i|}^2 & \cdots & t_{|\mathbf{B}_i|}^d \end{bmatrix}.$$

It can be shown that the optimal polynomial coefficients have the following closed form:

$$\alpha_{\mathbf{B}_i} = (\mathbf{V}_{\mathbf{B}_i}^T \mathbf{V}_{\mathbf{B}_i})^{-1} \mathbf{V}_{\mathbf{B}_i}^T \mathbf{x}_{\mathbf{B}_i},$$

and can be computed in $O(knd^\omega)$ time where ω is the matrix-multiplication exponent [11]. Then using Theorem 1 and the gradient for polynomial regression, the Jacobian of $h_{\mathbf{B}_i}(\mathbf{x})$ with respect to $\mathbf{x}$ forms a blockwise sparse matrix:

$$\begin{aligned}\frac{\partial [h_{\mathbf{B}_i}]_j}{\partial x_l} &= \frac{\partial}{\partial x_l}\big(\langle \alpha_{\mathbf{B}_i}, [\mathbf{V}_{\mathbf{B}_i}^T]_j \rangle\big) = \frac{\partial}{\partial x_l}\big(\langle (\mathbf{V}_{\mathbf{B}_i}^T \mathbf{V}_{\mathbf{B}_i})^{-1} \mathbf{V}_{\mathbf{B}_i}^T \mathbf{x}_{\mathbf{B}_i}, [\mathbf{V}_{\mathbf{B}_i}^T]_j \rangle\big) \\ &= \frac{\partial}{\partial x_l} [\mathbf{V}_{\mathbf{B}_i}^T]_j^T (\mathbf{V}_{\mathbf{B}_i}^T \mathbf{V}_{\mathbf{B}_i})^{-1} \mathbf{V}_{\mathbf{B}_i}^T \mathbf{x}_{\mathbf{B}_i} \\ &= \begin{cases} \big[\mathbf{V}_{\mathbf{B}_i} (\mathbf{V}_{\mathbf{B}_i}^T \mathbf{V}_{\mathbf{B}_i})^{-1} [\mathbf{V}_{\mathbf{B}_i}^T]_j\big]_l & \text{if } \ell \in \mathbf{B}_i \\ 0 & \text{otherwise.} \end{cases}\end{aligned}$$

B 1D Piecewise Linear Regression.

While piecewise constant regression has significant applications, we require more flexible models for more complex data, such as streaming sensor data or stock prices.

A popular refinement in such cases is piecewise linearity. The setup is similar as described above, and the goal is to partition a 1D function into disjoint intervals, except that for each interval, we solve the standard linear regression problem by minimizing the ℓ_2 error. Note that as before, our approach assumes that we have access to an oracle that returns the optimal piecewise linear approximation

$h(\mathbf{x})$ for any input $\mathbf{x}$. Let $\Pi = \{\mathbf{B}_1, \dots \mathbf{B}_k\}$ be the collection of partitions that minimizes the objective function:

$$\min_{\mathbf{B}_1,\dots,\mathbf{B}_k} \sum_{i=1}^{k} \sum_{j \in \mathbf{B}_i} (x_j - \alpha_i - \beta_i t_j)^2$$

where $\mathbf{t} = [0, 1, \dots, |\mathbf{B}_i| - 1]^T$ indexes elements within each piece of the partition.

For a certain partition $\mathbf{B}_i$, the optimal linear regression coefficients α_i and β_i corresponding to $\mathbf{B}_i$ can be computed in closed form (we omit a detailed derivation since it is classical):

$$\alpha_i = \frac{\sum_{j \in \mathbf{B}_i} x_j - \beta_i \sum_{j \in \mathbf{B}_i} t_j}{|\mathbf{B}_i|}, \quad \beta_i = \frac{|\mathbf{B}_i| \sum_{j \in \mathbf{B}_i} x_j t_j - (\sum_{j \in \mathbf{B}_i} x_j)(\sum_{j \in \mathbf{B}_i} t_j)}{|\mathbf{B}_i| \sum_{j \in \mathbf{B}_i} t_j^2 - (\sum_{j \in \mathbf{B}_i} t_j)^2}.$$

One can again show [14] that the partitions (along with the optimal coefficients) can be computed in $O(nk)$ running time.

To derive the Jacobian of $h(\mathbf{x})$ with respect to x_j, we again leverage Theorem 1. Since the partitions are decoupled, the Jacobian assumes the following form:

$$\begin{aligned}
&\frac{\partial h_{\mathbf{B}_i}}{\partial x_j} = \frac{\partial}{\partial x_j}(\alpha_i \mathbb{1}_{\mathbf{B}_i} + \beta_i \mathbf{t}_{\mathbf{B}_i}) \\
&= \frac{1}{|\mathbf{B}_i|}\Big(1 - \sum_{l \in \mathbf{B}_i} t_l \cdot \frac{|\mathbf{B}_i| t_j - \sum_{l \in \mathbf{B}_i} t_l}{|\mathbf{B}_i| \sum_{l \in \mathbf{B}_i} t_l^2 - (\sum_{l \in \mathbf{B}_i} t_l)^2}\Big) \cdot \mathbb{1}_{\mathbf{B}_i} + \frac{|\mathbf{B}_i| t_j - \sum_{l \in \mathbf{B}_i} t_l}{|\mathbf{B}_i| \sum_{l \in \mathbf{B}_i} t_l^2 - (\sum_{l \in \mathbf{B}_i} t_l)^2} \cdot \mathbf{t}_{\mathbf{B}_i} \\
&= \frac{1}{|\mathbf{B}_i|}\mathbb{1}_{\mathbf{B}_i} + \frac{|\mathbf{B}_i| t_j - \sum_{l \in \mathbf{B}_i} t_l}{|\mathbf{B}_i| \sum_{l \in \mathbf{B}_i} t_l^2 - (\sum_{l \in \mathbf{B}_i} t_l)^2}\Big(\mathbf{t}_{\mathbf{B}_i} - \frac{\sum_{l \in \mathbf{B}_i} t_l}{|\mathbf{B}_i|}\mathbb{1}_{\mathbf{B}_i}\Big).
\end{aligned}$$

Notice that our Jacobian formulation itself only depends on the size of a partition and is independent of the values of the specific elements of $\mathbf{x}$. This allows for the pre-computation of the sub-matrices for various-block sizes. In practice, the Jacobian vector-product therefore, can be calculated in $O(n)$ running time and can be further sped up using parallel computing[2].

C Experimental Details

C.1 Denoising

Dataset We generate the synthetic datasets with a prior of piecewise constant/linear with additive Gaussian noise. The length of each data are length 50 and we add Gaussian noise with standard deviations $3e^{-2}$ and $2e^{-2}$ for piecewise constant and piecewise linear data respectively. We create 1000 data containing perturbed signals and ground signals for the training and 100 for the testing.

Architecture We use the following model architecture for the vanilla DAE and DAE with piecewise constant/linear regularizer.

```
self.encoder = nn.Sequential(
          nn.Linear(50, 30),
          nn.ReLU(),
          nn.Linear(30, 10),
          nn.ReLU(),
      )
self.decoder = nn.Sequential(
```

[2]In both the piecewise constant and linear cases above, the Jacobian ostensibly appears to be constant irrespective of the input. However, this is not true since the partition depends on $\mathbf{x}$ and is calculated during the forward pass.

```
            nn.Linear(10, 30),
            nn.ReLU(),
            nn.Linear(30, 50),
            nn.Sigmoid()
        )
```

We train the DAEs using mean squared error between the output and piecewise approximation of the input. For the regulariser, we compute the MSE loss between the vanilla DAE output and the DPA layer that generates a piecewise approximation of the input. We observe that the MSE loss of vanilla DAE and DAE with regularizer in piecewise constant/linear dataset are almost equivalent (Figure 2). However, the DAE with regularizer better enforces piecewise priors as seen in Figure 1.

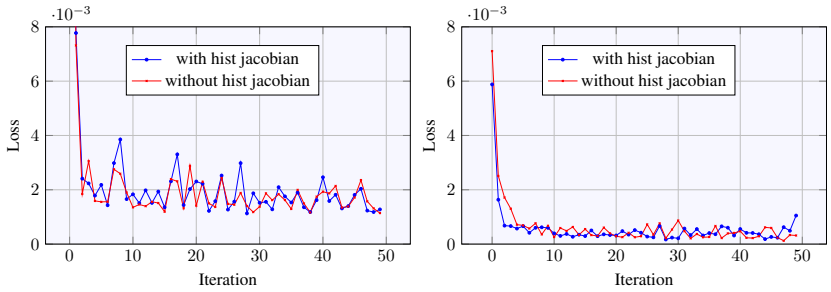


Figure 2: *Test loss comparison between vanilla DAE and DAE with regularizer. Both methods converge to nearly equal losses. However, regularized DAE better enforces the piecewise prior in terms of smoothness (fewer jagged or curved lines).*

C.2 Segmentation

Dataset Similar to [10], we use the Weizmann horse dataset for analysing the effect of DPA. The dataset consists of 378 images of single horses with varied backgrounds, and their corresponding ground truth. We divide the dataset into 80:10:10 ratio for training, validation and test respectively. Further, each image is normalized to a $[0, 1]$ domain by dividing it by 256.

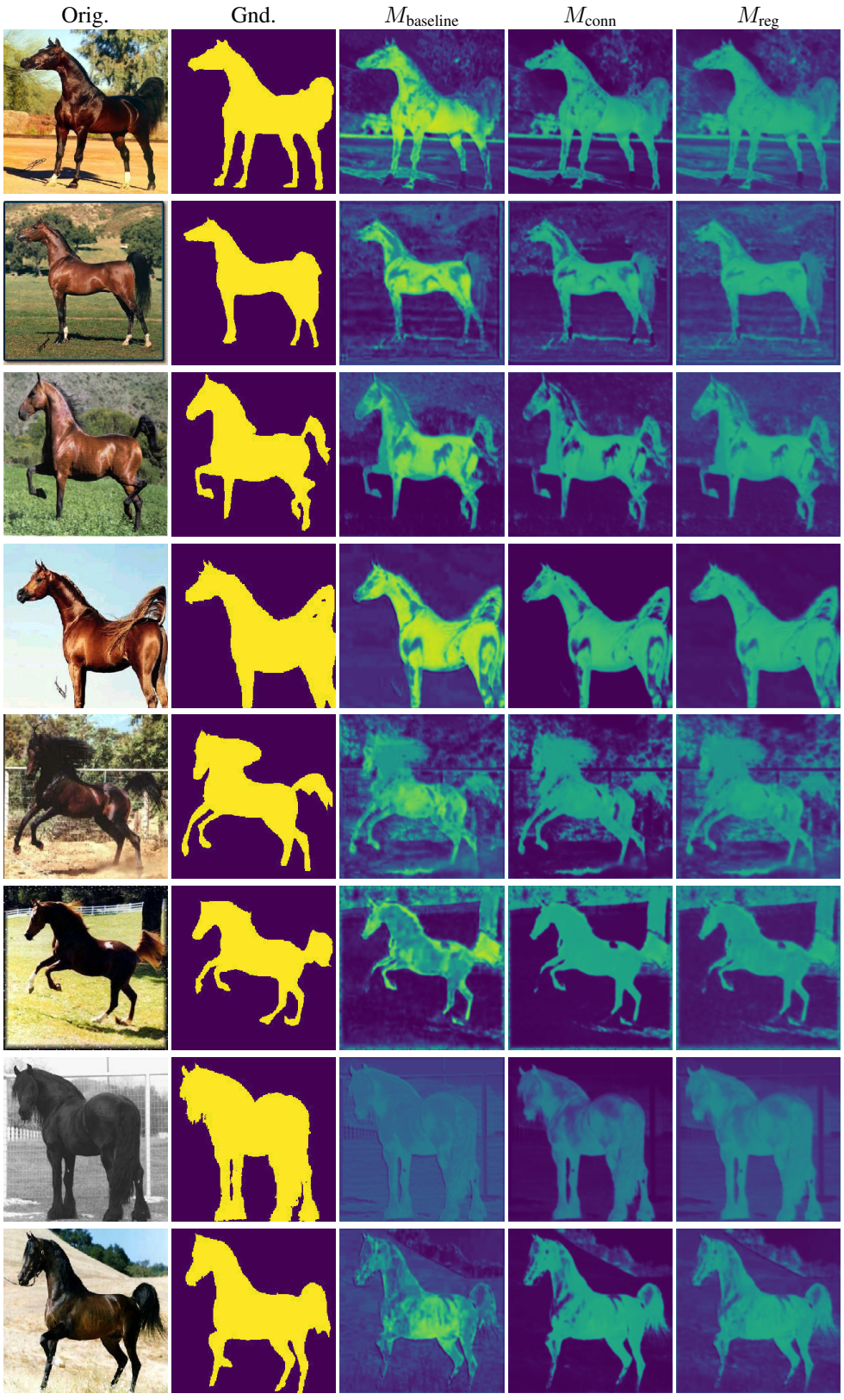
Architecture and Training. We use the following model architecture for training our segmentation networks.

```
self.layers = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 1, 3, padding=1),
            nn.Sigmoid()
        )
```

For M_{conn}, the DPA layer is appended after the sigmoid function. For the M_{reg} model, we pass the output of the above model through the DPA layer and minimize the sum of the MSE losses with respect to the ground truth, and the piecewise constant version of the output respectively. For the DPA layer, we use the `measure.label` function from Sci-Kit Image [20] to get the connected components. Since the function only works with integer valued images, we quantize the $[0, 1]$ float-valued output to a $[0, 10]$ integer valued image. Increasing the number of quantization bins improves results but also slows down the forward pass. We pick 10 for a good tradeoff between speed and accuracy.

For optimization, we use an ADAM optimizer with a learning rate of $3e^{-5}$ and a weight decay of $1e^{-4}$. All models are trained for $10,000$ epochs in order to ensure a fair comparison.

D Additional results



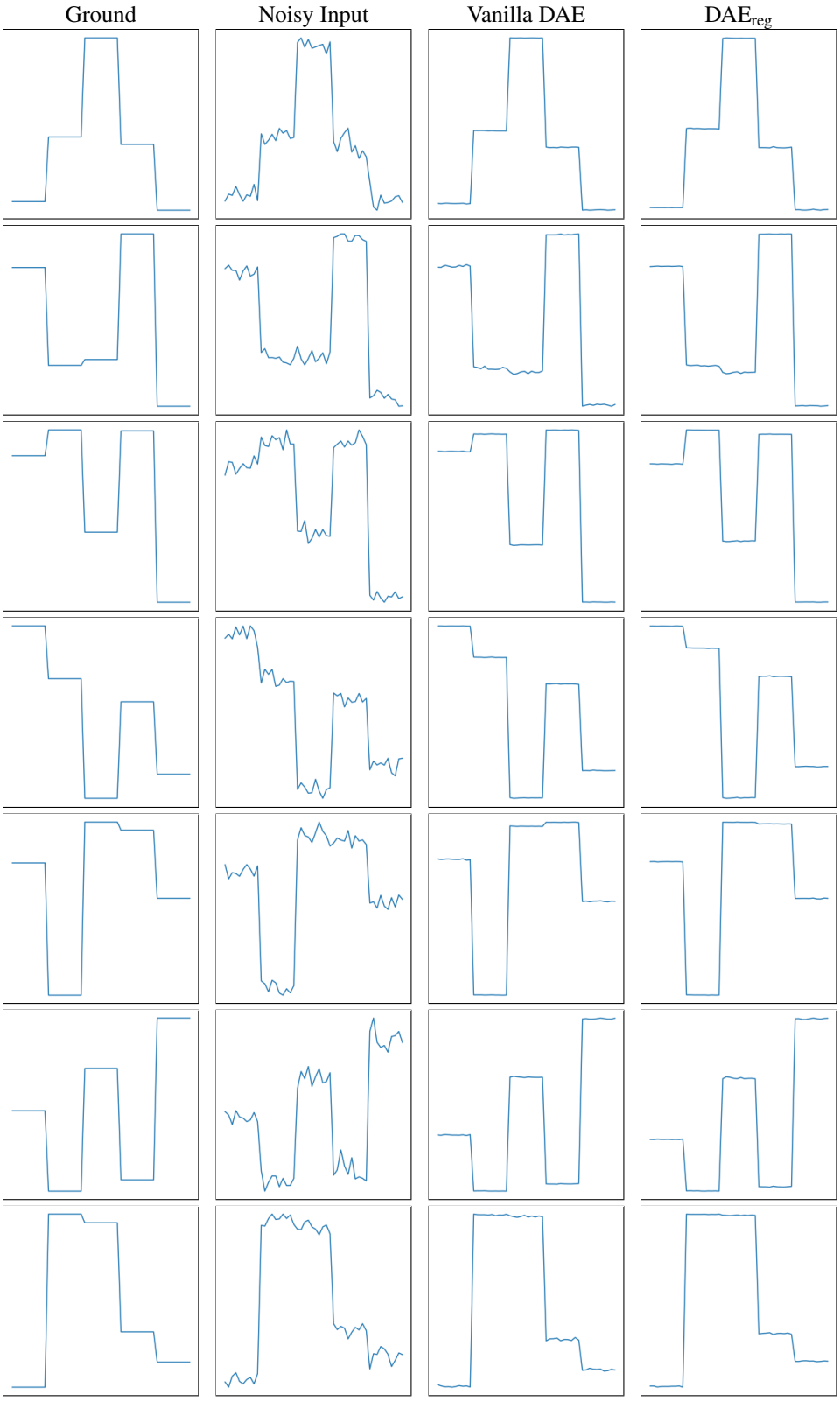


Figure 3: **DAE on piecewise constant prior signals.** Given perturbed signal (2nd column), each row demonstrates the visual comparison of denoised output between the vanilla DAE (3rd column) and DAE$_{reg}$ (4th column).

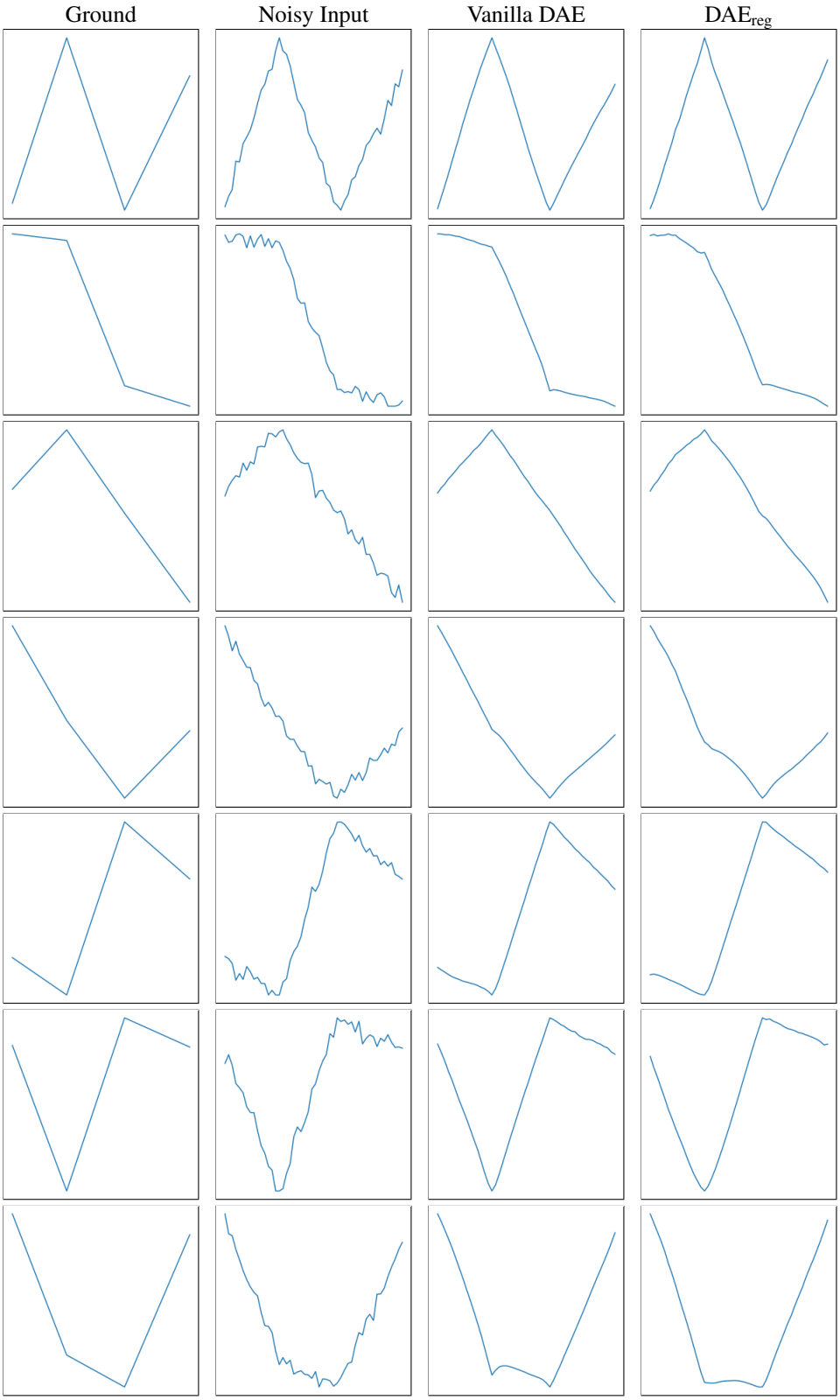


Figure 4: **DAE on piecewise linear prior signals.** Given perturbed signal (2nd column), each row demonstrates the visual comparison of denoised outputs between the vanilla DAE (3rd column) and DAE$_{reg}$ (4th column).