DIFFERENTIABLE PROGRAMMING FOR PIECEWISE POLYNOMIAL FUNCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

The paradigm of differentiable programming has significantly enhanced the scope of machine learning via the judicious use of gradient-based optimization. However, standard differentiable programming methods (such as autodiff) typically require that the machine learning models be differentiable, limiting their applicability. 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

Motivation: 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. This approach has found use in a large variety of applications such as scientific computing (Innes, 2020; Innes et al., 2019; Schafer et al., 2020), image processing (Li et al., 2018b), physics engines (Degrave et al., 2017), computational simulations (Alnæs et al., 2015), and graphics (Li et al., 2018a; Chen et al., 2019). One way to leverage differential programming modules is to encode additional structural priors as "layers" in a larger machine learning model. Inherent structural constraints such as monotonicity, or piecewise constancy, are particularly prevalent in applications such as physics simulations, graphics rendering, and network engineering; in such applications, it may help build models that obey such priors *by design*.

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 (Keogh et al., 2001). Similar piecewise approximations are also used in computer graphics for rendering (Kindlmann et al., 2003; Gross et al., 1995; Loop & Blinn, 2006), partial differential equation solvers (Hughes et al., 2005), network flow problems (Balakrishnan & Graves, 1989), and several other applications.

Challenges: To fully leverage differentiable programming for such problems, we would like to compute gradients "through" these approximation algorithms. However, algorithms for piecewise-polynomial function approximation often involve discontinuous (or even discrete) co-domains and may introduce undefined (or even zero) gradients. Generally, embedding such 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. This has led to recent advances in differentiable sorting (Blondel et al., 2020; Cuturi et al., 2019), dynamic programming (Mensch & Blondel, 2018), and submodular optimization (Djolonga & Krause, 2017). More recently, Deng et al. (2020) and Agrawal et al. (2019) propose methods to include convex optimization modules into deep networks. However, for discrete, non-differentiable objectives such as those encountered in piecewise regression, these approaches seem not applicable.

Our contributions: 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. This assumption holds true for a large class of real-world problems such as segmentation, speech diarization, and data summarization. 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. Our specific contributions are as follows:

1. We propose the use of piecewise polynomial function approximations as "layers" in differentiable programs, thereby extending their applicability to any setting with piecewise smooth outputs.
2. We derive efficient (nearly-linear time) algorithms for computing forward and backward passes for piecewise-polynomial fitting, in particular showing that the (weak) Jacobian can be represented using a *block sparse* matrix that can be efficiently used for backpropagation.
3. We show applications of our algorithm in piecewise polynomial approximation for denoising, training generative models with piecewise constraints, and image segmentation.

2 Related Work

The popularity of automatic differentiation has led several recent works to revisit standard elements in discrete algorithm design from a differentiable programming perspective. Here, we review a subset of related work in the literature[1].

Autodiff for discrete problems: Automatic differentiation (autodiff) algorithms enable gradient computations over basic algorithmic primitives such as loops, recursion, and branch conditions (Baydin et al., 2018). However, introducing more complex non-differentiable components requires careful treatment due to undefined or badly behaved gradients. For example, in the case of sorting and ranking operators, it can be shown that the corresponding gradients are either uninformative or downright pathological, and it is imperative the operators obey a 'soft' differentiable form. Cuturi et al. (2019) propose a differentiable *proxy* for sorting based on optimal transport. Blondel et al. (2020) improve this by proposing a more efficient differentiable sorting/ranking operator by appealing to isotonic regression. Berthet et al. (2020) introduce the use of stochastic perturbations to construct smooth approximations to discrete functions, and Xie et al. (2020); Lee et al. (2020) have used similar approaches to implement end-to-end trainable top-k ranking systems.

In a similar vein, Pogančić et al. (2020) approximate gradients with respect to combinatorial optimization solvers by constructing linear approximations to discrete-valued functions. Amos & Kolter (2019); Agrawal et al. (2019) define backpropagation over convex optimization problems by factorizing over interior point solvers. Mensch & Blondel (2018) estimate gradients over dynamic programs by relaxing the (non-differentiable) *max* operator using strongly convex regularizers.

Structured priors as neural "layers": As mentioned above, one motivation for our approach arises from the need for enforcing structural priors for scientific computing applications. Encoding non-differentiable priors such as the solutions to specific partial differential equations (Sheriffdeen et al., 2019), geometrical constraints (Joshi et al., 2020; Chen et al., 2019), and spatial consistency measures (Djolonga & Krause, 2017) typically require massive amounts of structured training examples. On the other hand, our approach directly produces a differentiable method for general families of piecewise polynomial priors.

Piecewise function approximation: Piecewise polynomial regression has been separately studied in several scientific and algorithmic sub-fields. Our method is agnostic to the specific regression algorithm used and enables the use of several fast approaches (Jagadish et al., 1998; Acharya et al., 2015; Magnani & Boyd, 2009; Lemire, 2007) within deep neural networks. This allows us to expand the scope of general differentiable programs to tasks such as data summarization, streaming, and segmentation.

[1]While tricks such as straight-through gradient estimation (Bengio, 2013) may also be applicable, they are heuristic in nature and may be inaccurate for specific problem instances (Yin et al., 2019).

3 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 $h \in \mathbb{R}^n$ is said to be k-piecewise polynomial with degree d if h is the sum of sub-functions $h = \sum_{i=1}^{k} h_{\mathbf{B}_i}$, where $\Pi_{\mathbf{B}} = \{\mathbf{B}_1, \ldots, \mathbf{B}_k\}$ is a partition of $[n]$, and $h_{\mathbf{B}_i}$ is a degree-d polynomial. We denote $\mathcal{F}_d$ as the family of all such piecewise polynomial functions $h \in \mathbb{R}^n$.

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 (as parameterized by B_i) by solving the optimization problem:

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

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,*

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

We derive this by substituting the definition of $h(\mathbf{x})$ in the above expression, and using the fact that the indicator function is non-zero only for elements belonging to a partition, $\mathbf{B}_i$. Note that this derivation holds for any piecewise partitioning function as long as the sub-functions themselves are differentiable. However, in this work, we specifically focus on specific cases that are often used in machine learning.

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.

Formally, consider a piecewise constant function $h : \mathbb{R}^n \to \mathbb{R}^n$ with at most k interval partitions; in the data streaming literature, such a function is called a *k-histogram*. Since the best (in terms of ℓ_2-norm) constant approximation to a function is its mean, 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 $[\mathbf{x}_{\mathbf{B}_i}]_j = [\mathbf{x}]_j \mathbb{1}(j \in \mathbf{B}_i)$. In other words, 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 (Jagadish et al., 1998)). The running time of such approaches is typically $O(nk)$, which is constant for fixed k; see Acharya et al. (2015) for a more detailed treatment.

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

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

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 $\mathbf{J} \in \mathbb{R}^{n\times n}$:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{J}_k \end{bmatrix}$$

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.

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 (Jagadish et al., 1998) 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].

Generalization to 1D piecewise polynomial fitting: We now derive differentiable forms of generalized piecewise d-polynomial regression, which is used in applications such as spline fitting.

As before, $h : \mathbb{R}^n \to \mathbb{R}^n$ is 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. We get a similar closed-form expression for the coefficients as in Section 3. 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 (Guha et al., 2006). 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:

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

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.

4 2D PIECEWISE CONSTANT FUNCTIONS

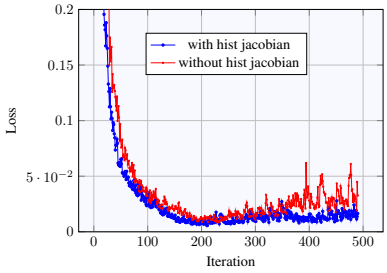
Our approach can be (heuristically) extended to 2D data with minor modifications. To illustrate this, we consider the problem of image segmentation, which can be viewed as representing the domain of an image into a disjoint union of subsets. Neural-network based segmentation involves training a model (deep or otherwise) to map the input image to a segmentation map, which is a piecewise constant function. However, standard neural models trained in a supervised manner with image-segmentation map pairs would generate pixel-wise predictions, which could lead to disconnected regions (or holes) as predictions. We leverage our approach to enforce deep models to predict piecewise constant segmentation maps. In 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 (Wu et al., 2005)) 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

[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.

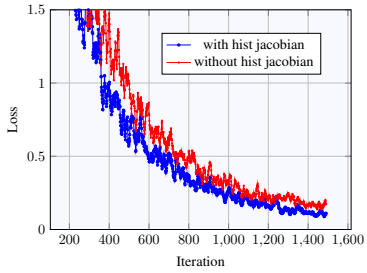
of the block Jacobian where each partition is now represented as a channel which is only non-zero in the positions corresponding to the channel. Formally, if the image $\mathbf{x} \in \mathbb{R}^n$ is represented as the union of k partitions, $h(\mathbf{x}) = \bigcup_{i=1}^{k} \mathbf{B}_i$, the Jacobian, $\mathbf{J}(\mathbf{x}) = \partial h(\mathbf{x})/\partial \mathbf{x} \in \mathbb{R}^{n \times n}$ and,

$$(\mathbf{J})_{ij} = \begin{cases} 1/|\mathbf{B}_k| \text{ if } h(\mathbf{x})_i \in \mathbf{B}_k, \\ 0 \text{ otherwise.} \end{cases} \tag{2}$$

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.



Data-free piecewise *constant* $k = 3$



Data-free piecewise *linear* $k = 3$

Figure 1: *Loss comparison for data-free training of generative models for piecewise constant functions. For the loss* $L = \|G_\theta(z) - h(G_\theta(z))\|_2^2$ *where* G *and* h *are the generator and histogram layer respectively, a blue plot uses both our forward and backward passes, while a red plot does not leverage Jacobian information by using the straight-through estimator. We observe that using gradient information from the histogram layer enables improved optimization.*

5 Experiments

Generative models for 1D signals: We first analyze the performance of our approach in toy cases, leveraging the piecewise constant/linear layers to train a generative model for synthetic piecewise polynomial signals. Denoting $\mathbf{z}$ as a Gaussian noise vector as input and G_θ as a generative model parameterized by θ, we train the model with mean-square error (MSE) loss between generated output $(G_\theta(\mathbf{z}))$ and its piecewise polynomial approximation $(h(G_\theta(\mathbf{z})))$: $L_{\text{MSE}} = \|G_\theta(\mathbf{z}) - h(G_\theta(\mathbf{z}))\|_2^2$. Since this loss can result in mode collapse, we adopt the additional loss maximizing the distance of generated output from different noise inputs: $L_{\text{latent}} = \|\mathbf{z}_1 - \mathbf{z}_2\|_2^2 / \|G_\theta(\mathbf{z}_1) - G_\theta(\mathbf{z}_2)\|_2^2$. We update parameters θ from the L_{MSE} and L_{latent} in an alternating fashion.

We conduct four different generators: (1) piecewise constant with $k = 3$, (2) piecewise constant with $k = 10$, (3) piecewise linear with $k = 3$, (4) piecewise linear with $k = 5$. Furthermore, to verify the utility of the Jacobian for the piecewise polynomial layer, we compare with the *no-Jacobian* case by using a straight-through gradient estimator. We defer additional details to the Appendix. Figure 2 and Figure 3 compare the generated output of piecewise constant/linear signals respectively. We observe that the generated signals from the model learned with Jacobian from piecewise layers are smoother than outputs without histogram layer Jacobian. Figure 1 presents comparison plots of the L_{MSE} of above four experiment setups. Overall, we observe improvement in performance using our differentiable piecewise polynomial approximation layers.

1D signal denoising: Next, 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 $\frac{1}{2}\|f(\mathbf{x}) - \mathbf{y}\|_2^2$ and (2) a DAE along with a piecewise constant layer with additional MSE loss $\frac{1}{2}\|f(\mathbf{x}) - h(f(\mathbf{x}))\|_2^2$. We train the DAE with the synthetic dataset with piecewise constant/linear priors with additive Gaussian noise.

We observe that adding our piecewise regularizer with a histogram layer further smooths out the noisy input compared with vanilla DAE. Figure 4 compares the denoised inputs between DAE

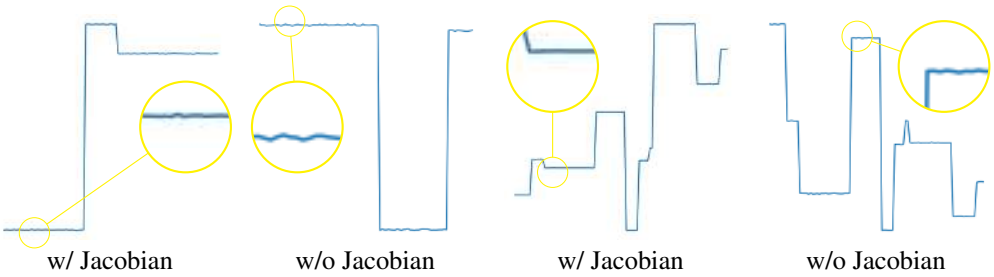


Figure 2: ***Results for 1D piecewise constant function generation.*** *Generators updated with DPA layer Jacobians generates smoother lines compared to generators trained with straight-through gradient backpropagation for the backward pass.*

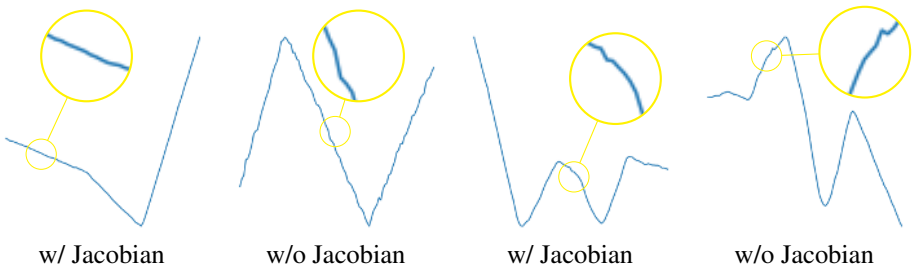


Figure 3: ***Results for data-free training of generative models for 1D piecewise linear functions.*** *Similar to piecewise constant case, the generator trained with the backward pass from histogram layer generate smoother images than with the straight-through estimator.*

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

and DAE_{reg}. While both DAE and DAE_{reg} mostly achieves near-correct partitions, DAE_{reg} further smooths the denoised input compare to the vanilla version. We defer the detailed experiment setup to the Appendix.

Image segmentation: For the 2D piecewise regression approach presented in Section 4, we consider the problem of image segmentation using deep neural networks. We use a similar setup as in Djolonga & Krause (2017) 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 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 (Borenstein & Ullman, 2004) 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).

We observe that our piecewise approximation layer provides more consistent segmentation maps with fewer holes for both M_{conn} and M_{reg}. (see Figure 5). Additionally, we see that our proposed models perform better than the baseline model in Jaccard scores with respect to the ground truth. Specifically, M_{reg} outperforms all the other settings by a significant amount. We also observe that the effect of our proposed approach is more pronounced for images with smaller partitions. This is to be expected, since the expression for the Jacobian contains the inverse of the partition size, thereby ensuring a better estimate of the descent direction than a straight through estimator for smaller

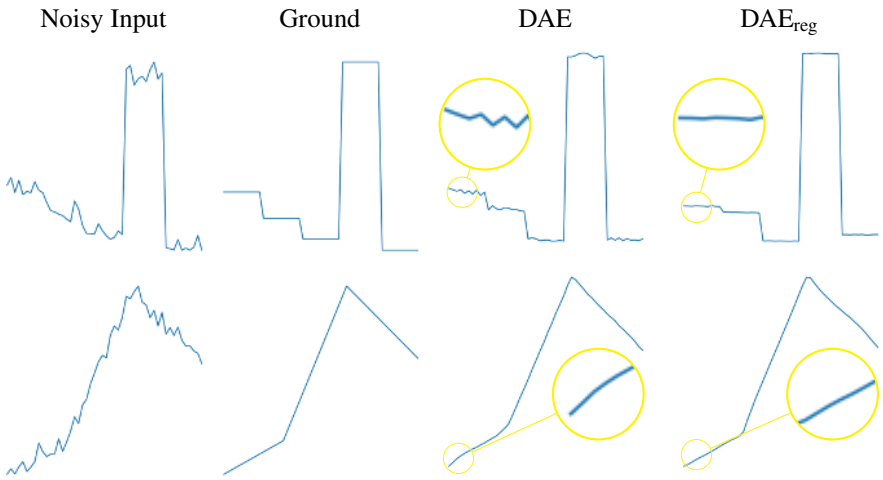


Figure 4: *Denoising results of 1D piecewise constant/linear signals. We observe that training DAE with a regularizer with piecewise layer denoises the perturbed input signals better than vanilla DAE.*

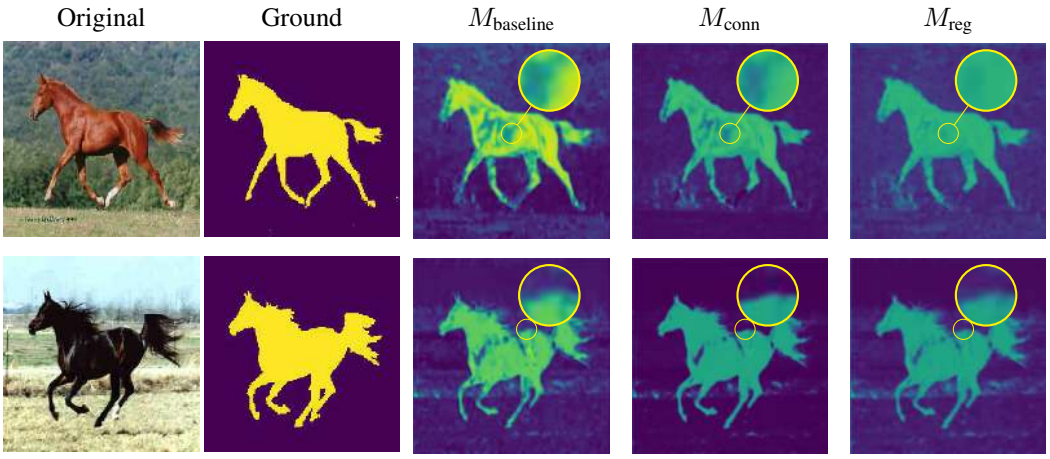


Figure 5: ***Segmentation results.*** *The two models, M_{baseline} and M_{conn} were trained with and without the differentiable connected component layer respectively. M_{reg} minimizes the difference between the network output and its piecewise approximation along with the standard segmentation error. 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.*

partitions. This would be very useful for medical image segmentation tasks, where the objects are generally small compared to the original image size.

6 Discussion and Conclusion

We introduce a principled approach to estimate gradients with respect to piecewise polynomial function approximations. Specifically, we derive the (weak) Jacobian in the form of a block-sparse matrix based on the partitions generated by any polynomial approximation algorithm (which serves as the forward pass). The block structure allows for fast computation of the backward pass, thereby extending the application of differentiable programs (such as deep neural networks) to tasks involving piecewise polynomial priors, such as denoising and image segmentation. Our approach can be extended to higher-dimensional functions provided appropriate piecewise partitioning routines are available. Extension of our approach to more general families of piecewise differentiable function classes is a promising direction for future work.

REFERENCES

Jayadev Acharya, Ilias Diakonikolas, Chinmay Hegde, Jerry Zheng Li, and Ludwig Schmidt. Fast and near-optimal algorithms for approximating distributions by histograms. In *Proc. ACM SIGMOD-SIGACT-SIGAI Symp. on Principles of Database Systems*, 2015.

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.

Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. *ArXiv*, 1703.00443, 2019.

Anantharam Balakrishnan and Stephen C Graves. A composite algorithm for a concave-cost network flow problem. *Networks*, 19(2):175–202, 1989.

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Machine Learning Research*, 18(153): 1–43, 2018. URL http://jmlr.org/papers/v18/17-468.html.

Yoshua Bengio. Estimating or propagating gradients through stochastic neurons. *ArXiv*, abs/1305.2982, 2013.

Quentin Berthet, Mathieu Blondel, O. Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis R. Bach. Learning with differentiable perturbed optimizers. *ArXiv*, abs/2002.08676, 2020.

Mathieu Blondel, O. Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. *ArXiv*, abs/2002.08871, 2020.

Eran Borenstein and Shimon Ullman. Learning to segment. In *Euro. Conf. Comp. Vision*, pp. 315–328. Springer, 2004.

W. Chen, Jun Gao, Huan Ling, Edward 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.

Marco Cuturi, O. Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019.

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

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. CvxNet: Learnable convex decomposition. In *IEEE Conf. Comp. Vision and Pattern Recog*. IEEE, 2020.

Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, pp. 1013–1023, 2017.

Markus H Gross, Lars Lippert, A Dreger, and R Koch. A new method to approximate the volume-rendering equation using wavelet bases and piecewise polynomials. *Computers & Graphics*, 19 (1):47–62, 1995.

Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. on Database Systems (TODS)*, 31(1):396–438, 2006.

Thomas JR Hughes, John A Cottrell, and Yuri Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comp. methods in Applied Mechanics and Engineering*, 194(39-41):4135–4195, 2005.

Michael J. Innes. Algorithmic differentiation. In *Proc. Conf. ML. and Systems. (MLSys)*, 2020.

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

H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *Proc. of Int. Conference on Very Large Data Bases (VLDB)*, 1998.

Ameya Joshi, Minsu Cho, Viraj Shah, B. Pokuri, Soumik Sarkar, Baskar Ganapathysubramanian, and Chinmay Hegde. Invnet: Encoding geometric and statistical invariances in deep generative models. In *AAAI*, 2020.

Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Proc. of IEEE Int. Conf. on data mining*, pp. 289–296. IEEE, 2001.

Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Moller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization, 2003. VIS 2003.*, pp. 513–520. IEEE, 2003.

Hyunsung Lee, Yeongjae Jang, Jaekwang Kim, and Honguk Woo. A differentiable ranking metric using relaxed sorting opeartion for top-k recommender systems. *ArXiv*, abs/2008.13141, 2020.

D. Lemire. A better alternative to piecewise linear time series segmentation. In *SDM*, 2007.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable Monte-Carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6):1–11, 2018a.

Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. Differentiable programming for image processing and deep learning in halide. *ACM Trans. Graph.*, 37 (4):1–13, 2018b.

Charles Loop and Jim Blinn. Real-time GPU rendering of piecewise algebraic surfaces. In *SIGGRAPH*, pp. 664–670. ACM, 2006.

Alessandro Magnani and Stephen P Boyd. Convex piecewise-linear fitting. *Optimization and Engineering*, 10(1):1–17, 2009.

A. Mensch and Mathieu Blondel. Differentiable dynamic programming for structured prediction and attention. *ArXiv*, abs/1802.03676, 2018.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *Proc. Int. Conf. Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

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

Sheroze Sheriffdeen, J. Ragusa, J. Morel, M. Adams, and T. Bui-Thanh. Accelerating PDE-constrained inverse solutions with deep learning and reduced order models. *ArXiv*, abs/1912.08864, 2019.

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 2014.

Kesheng Wu, Ekow Otoo, and Arie Shoshani. Optimizing connected component labeling algorithms. In *Medical Imaging 2005: Image Processing*, volume 5747, pp. 1965–1976. International Society for Optics and Photonics, 2005.

Yujia Xie, Hanjun Dai, M. Chen, Bo Dai, Tuo Zhao, H. Zha, Wei Wei, and T. Pfister. Differentiable top-k operator with optimal transport. *ArXiv*, abs/2002.06504, 2020.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. In *Proc. Int. Conf. Learning Representations (ICLR)*, 2019.

A PROOFS AND DERIVATIONS

Proof for Theorem 1

Proof. The proof follows similar arguments as in Proposition 4 from Blondel et al. (2020).

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.

□

B EXPERIMENTAL DETAILS

B.1 DATA-FREE GENERATIVE MODEL TRAINING

Architecture We use the following architecture concatenated with fully-connected layers, represented in PyTorch notation.

```
self.layers = nn.Sequential(
          nn.Linear(latent_space, 20),
          nn.LeakyReLU(0.1),
          nn.Linear(20, 60),
          nn.LeakyReLU(0.1),
          nn.Linear(60, 100),
      )
```

While the data-free generator model has two different loss $L_{\text{MSE}} = \|G_{\theta(\mathbf{z})} - h(G_\theta(\mathbf{z}))\|_2^2$ and $L_{\text{latent}} = \|\mathbf{z}_1 - \mathbf{z}_2\|_2^2 / \|G_\theta(\mathbf{z}_1) - G_\theta(\mathbf{z}_2)\|_2^2$, we update the parameter θ alternatively between L_{MSE} and L_{latent}. We use an ADAM optimizer with a learning rate of 0.001 for both L_{MSE} and L_{latent}.

B.2 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(
```

```
            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 6). However, the DAE with regularizer better enforces piecewise priors as seen in Figure 4.

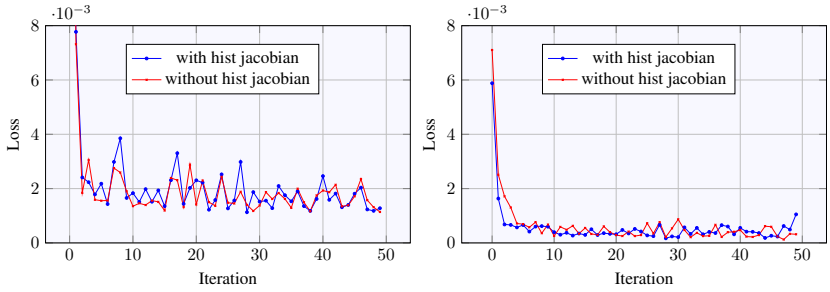


Figure 6: *Test loss comparison between vanilla DAE and DAE with regularizer. Both methods converge to nearly equal losses. However, regularized DAE better enforces the piecewise prior in terms of smoothness (fewer jagged or curved lines).*

B.3 SEGMENTATION

Dataset Similar to Djolonga & Krause (2017), 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.

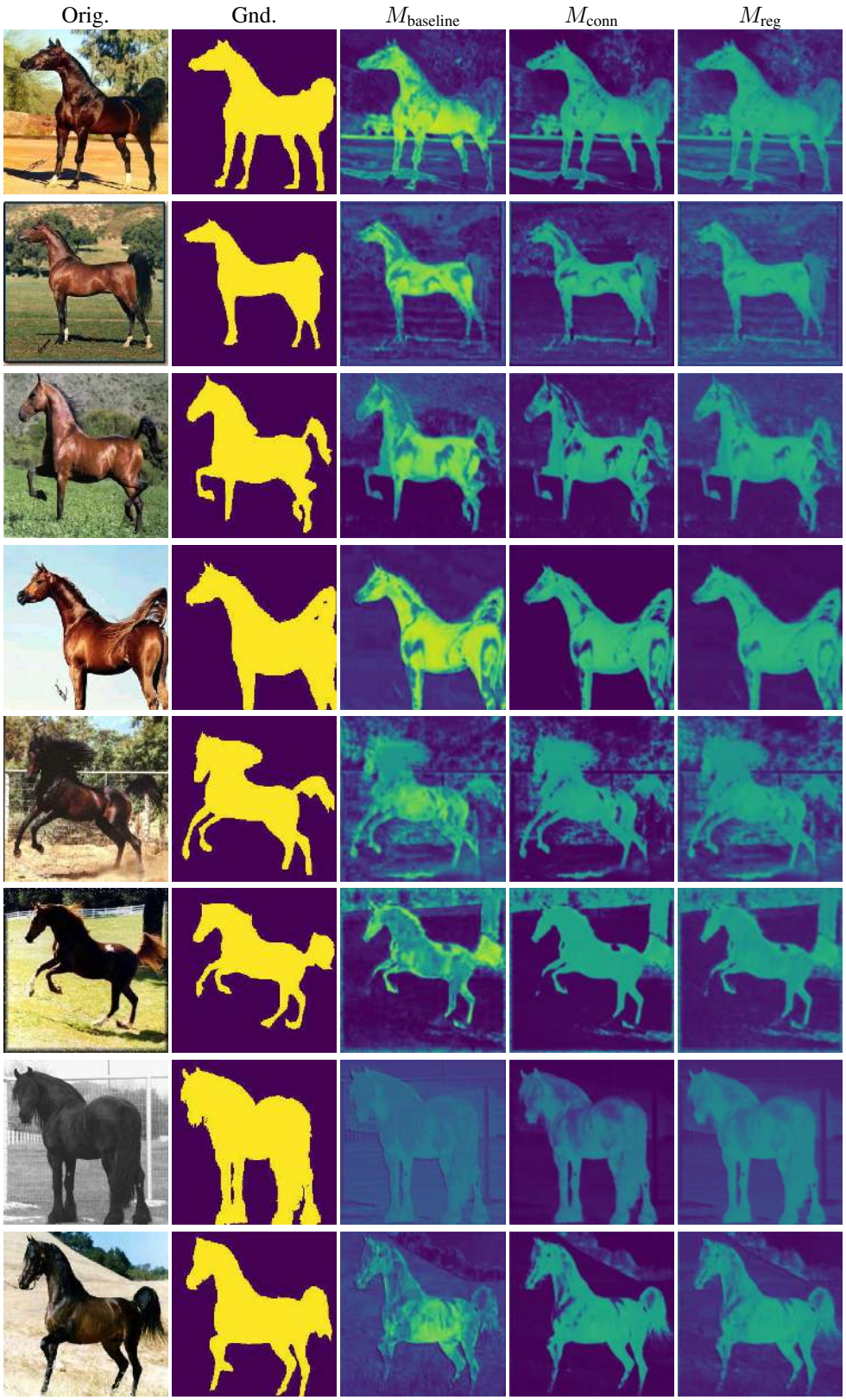
Architecture and Training. We use the following model architecture for training our segmentation networks.

```
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 (van der Walt et al., 2014) 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.

C ADDITIONAL RESULTS



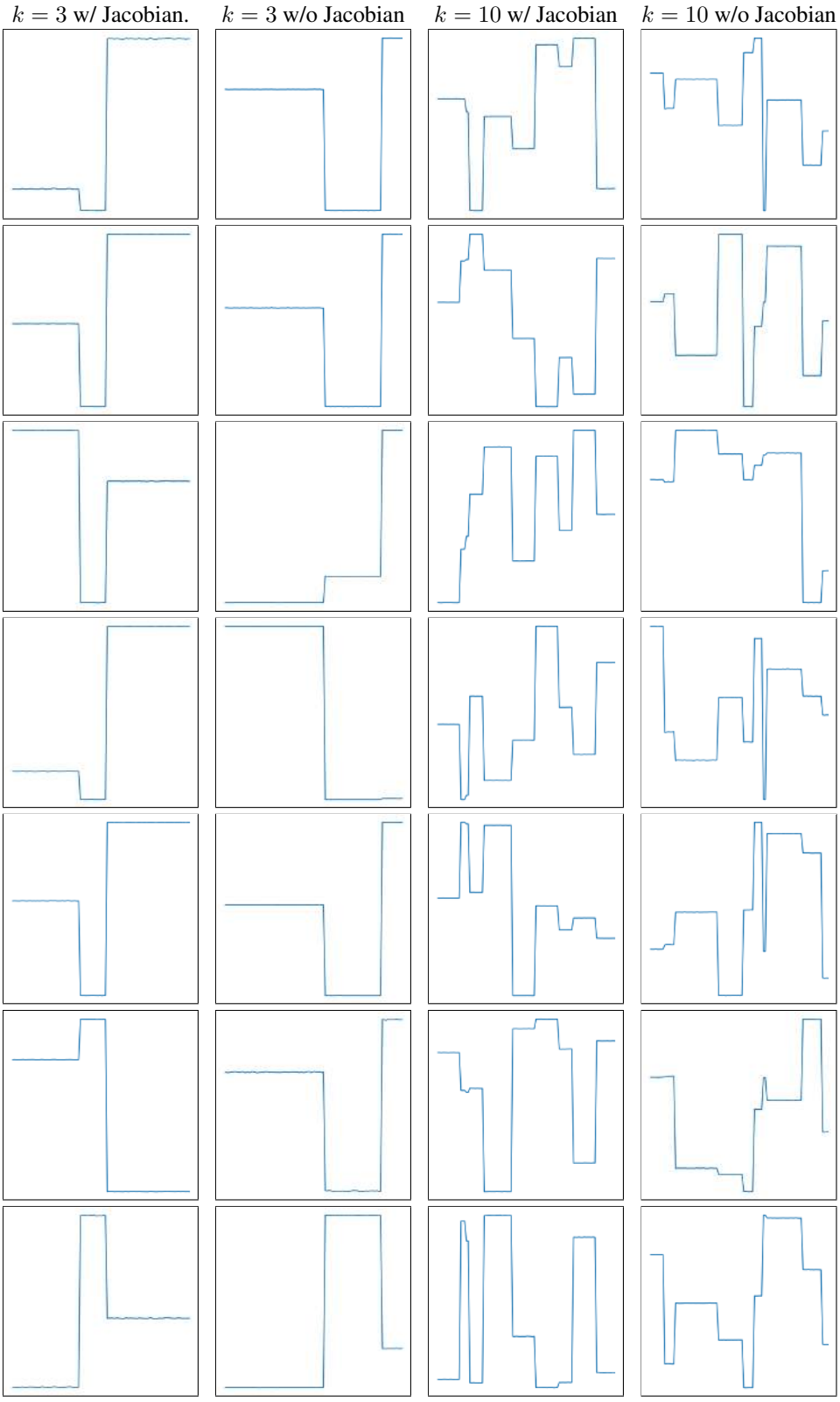


Figure 7: **Generating Piecewise Constant Vectors.** Data-free training of a neural network with the backward pass from DPA layer allows us to generate smoother linear segments than without the backward pass.

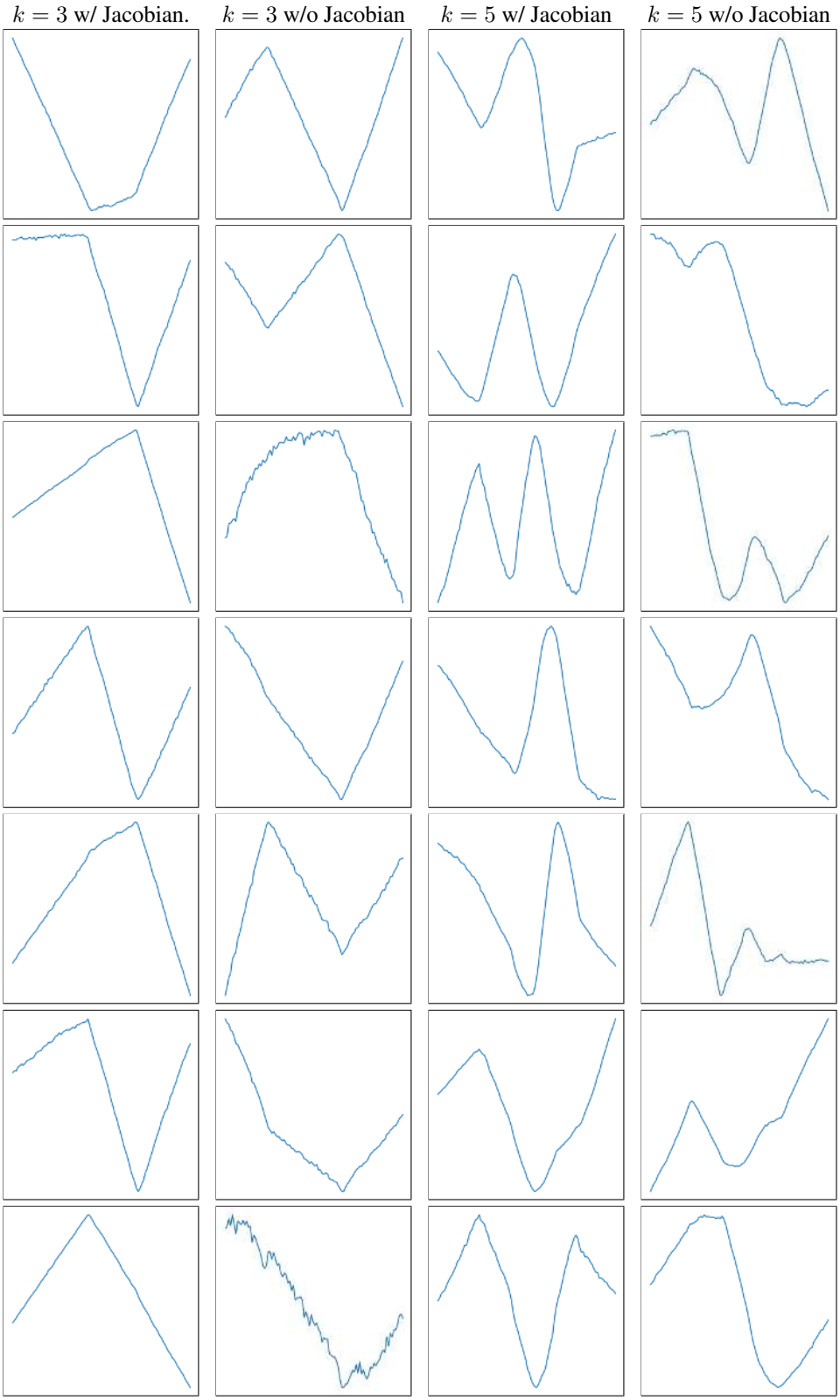


Figure 8: **Generating Piecewise Linear Vectors.** Data-free training of a neural network with the backward pass from DPA layer allows us to generate smoother linear segments than without the backward pass.

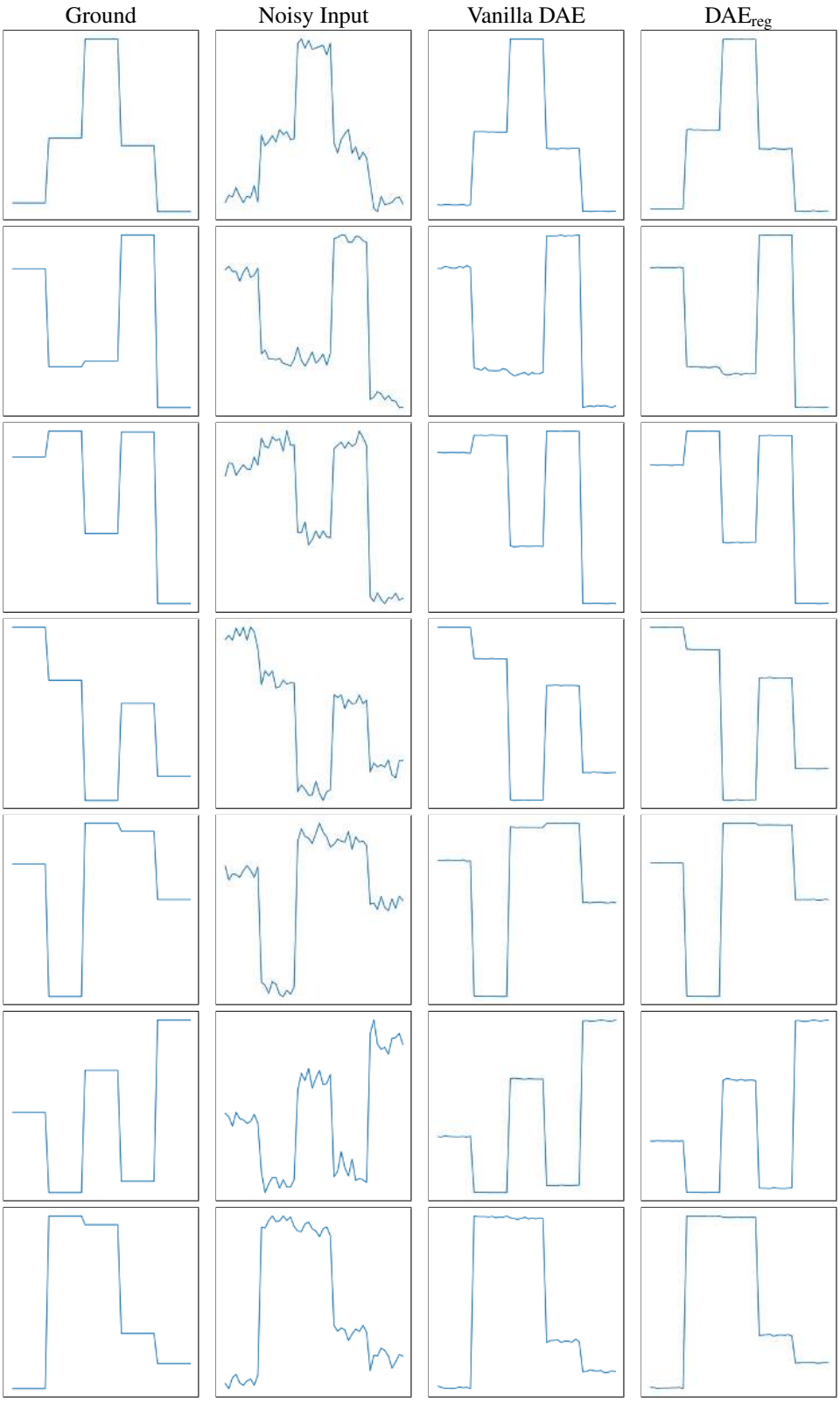


Figure 9: **DAE on piecewise constant prior signals.** Given perturbed signal (2nd column), each row demonstrates the visual comparison of denoised output between the vanilla DAE (3rd column) and DAE$_{reg}$ (4th column).

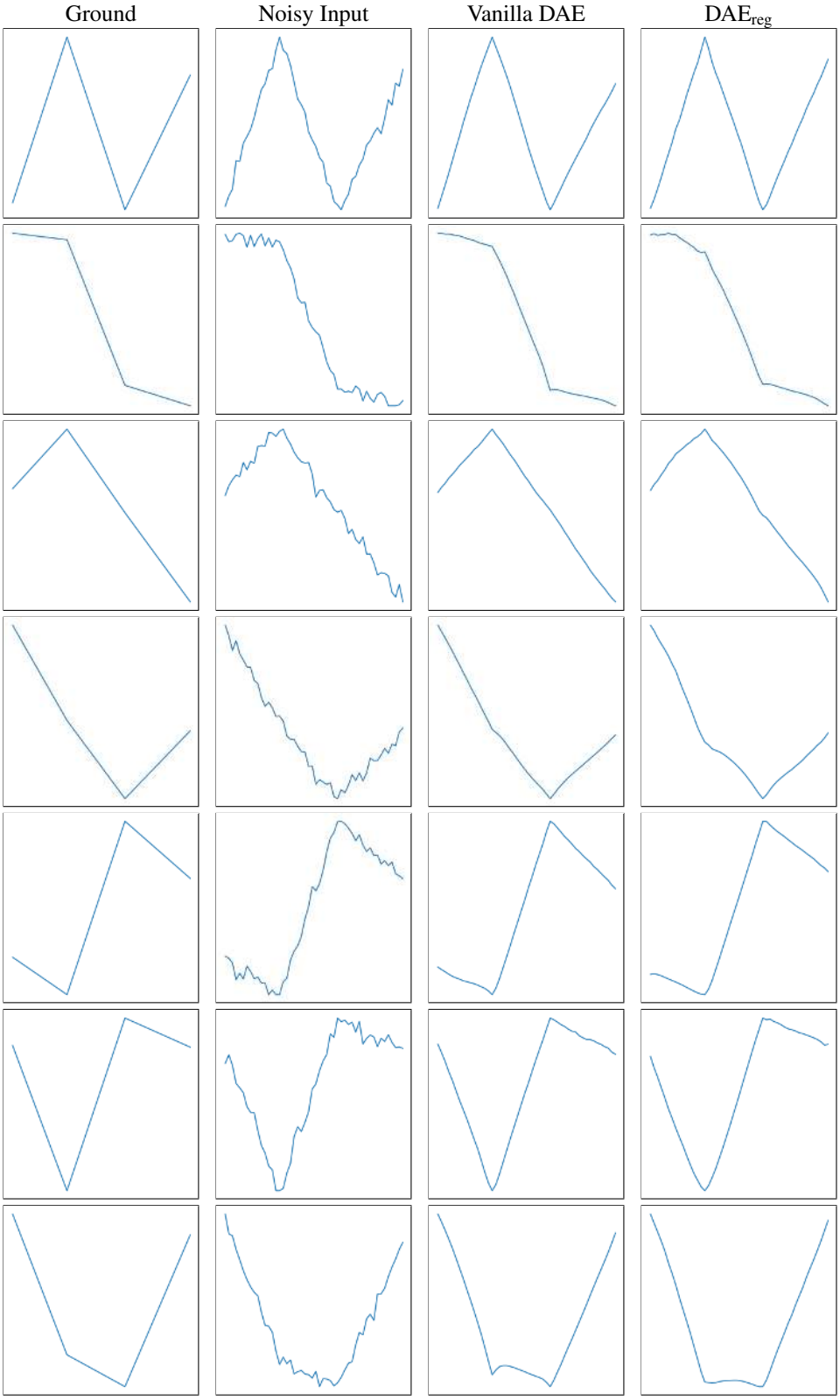


Figure 10: **DAE on piecewise linear prior signals.** Given perturbed signal (2nd column), each row demonstrates the visual comparison of denoised outputs between the vanilla DAE (3rd column) and DAE$_{reg}$ (4th column).