Hyper Input Convex Neural Networks for Shape Constrained Learning and Optimal Transport

Shayan Hundrieser[1]* **Insung Kong**[2]*

Johannes Schmidt-Hieber[3]

[1]Department of Statistics and Data Science, Yale University, New Haven, CT, USA
[2]Department of Mathematical Sciences, UNIST, Ulsan, South Korea
[3]Humboldt-University of Berlin and Weierstrass Institute, Berlin, Germany
shayan.hundrieser@yale.edu, kong@unist.ac.kr, schmidt-hieber@wias-berlin.de

Abstract

We introduce Hyper Input Convex Neural Networks (HyCNNs), a novel neural network architecture designed for learning convex functions. HyCNNs combine the principles of Maxout networks with input convex neural networks (ICNNs) to create a neural network that is always convex in the input, theoretically capable of leveraging depth, and performs reliably when trained at scale compared to ICNNs. Concretely, we prove that HyCNNs require exponentially fewer parameters than ICNNs to approximate quadratic functions up to a given precision. Throughout a series of synthetic experiments, we demonstrate that HyCNNs outperform existing ICNNs and MLPs in terms of predictive performance for convex regression and interpolation tasks. We further apply HyCNNs to learn high-dimensional optimal transport maps for synthetic examples and for single-cell data, where they oftentimes outperform ICNN-based neural optimal transport methods and other baselines across a wide range of settings.

1 Introduction

This paper introduces a novel neural network architecture, called Hyper Input Convex Neural Network (HyCNN), which (i) is always convex in its inputs, (ii) is explicitly designed to harness depth for approximation, and (iii) outperforms alternative procedures in terms of predictive power when trained at depth for large data sets, see Figure 1.

Shape constrained learning. Among shape constraints, *convexity in the input* is favorable for multiple reasons. First, convexity goes beyond being a convenient inductive bias as it is intrinsic to several core problems. In optimal transport (OT) with squared Euclidean costs, Brenier's theorem asserts that the OT map between absolutely continuous probability measures with finite second moment is given by the gradient of a convex potential (Brenier, 1991; Villani, 2009; Peyré & Cuturi, 2019). Hence, learning an OT map is naturally linked to learning a convex function (Taghvaei & Jalali, 2019; Makkuva et al., 2020; Bunne et al., 2023). In invertible generative modeling, gradients of (strongly) convex potentials yield monotone diffeomorphisms that serve to define normalizing flows (Rezende & Mohamed, 2015; Huang et al., 2021). Convex models also arise in optimal control, where convexity is used to ensure planning computational feasibility while using expressive learned dynamics or cost surrogates (Chen et al., 2019).

*Equal contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

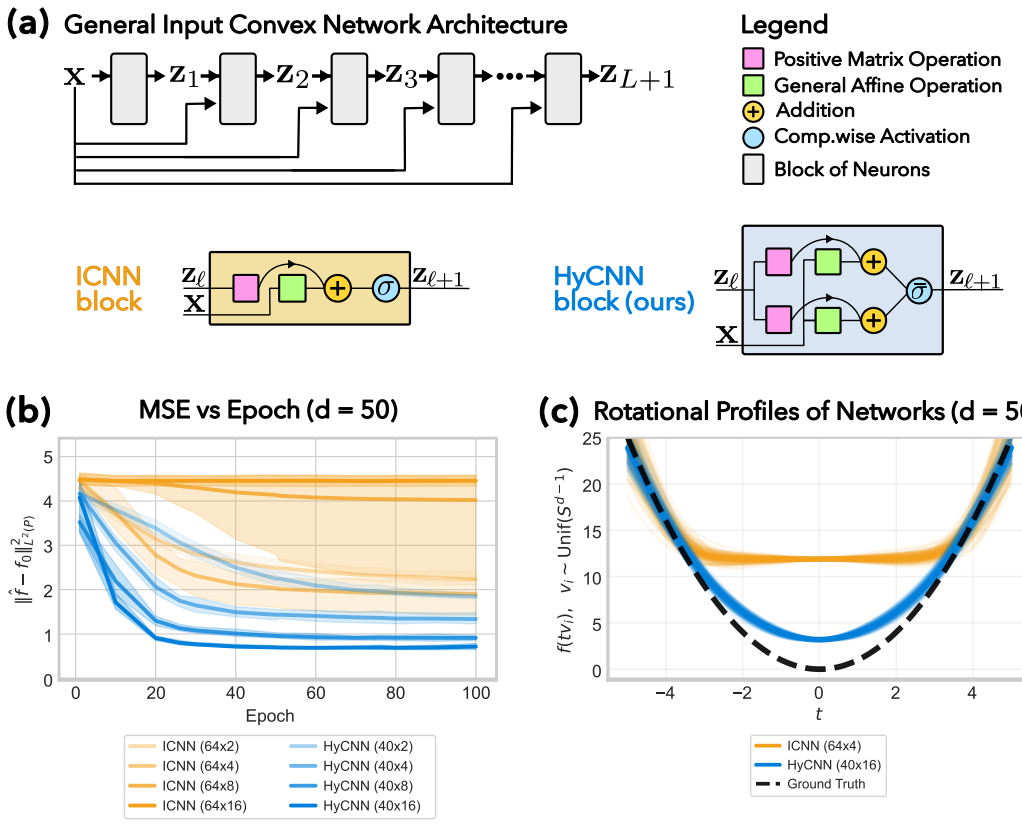

Figure 1: Input convex neural networks (ICNNs) versus Hyper Input Convex Neural Networks (HyCNNs). **(a)** Depiction of generic input convex network architecture. ICNN blocks comprise one lane, whereas Hyper ICNN (HyCNN) blocks involve two lanes of matrix operations. **(b)** Averaged Empirical MSE over 10 repetitions with $(10\% - 90\%)$ confidence bands for learning the function $f_0(x) = \|\mathbf{x}\|_2^2$ on $\mathbb{R}^d$ with $d = 50$ based on $n = 5{,}000$ samples of the form $(\mathbf{X}_i, Y_i)_{i\in[n]}$ with $\mathbf{X}_i \sim \mathrm{Unif}[-1,1]^d$ and $Y_i = f_0(\mathbf{X}_i) + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0,1)$. Two network architectures are used, ICNNs (Amos et al., 2017) and HyCNNs (ours) where the label $(m \times L)$ indicates width m and depth L of the network. **(c)** Rotational profiles $t \mapsto f(t\mathbf{v}_i)$ of ground truth (dashed) and learned networks with best MSE performance within network class, for 100 randomly drawn directions $\mathbf{v}_i \sim \mathrm{Unif}(\mathbb{S}^{d-1})$. Choosing width 64 for ICNN and 40 for HyCNN yields near equal parameter-counts per depth.

Second, for a real-valued function $f(\cdot)$ that is convex and coercive, operations such as

$$\arg\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$$

become tractable via convex optimization, and gradients/subgradients are globally well-behaved. As such, network architectures that are convex in the input are natural candidates for learning energy functions, which are ubiquitous in machine learning and statistics, e.g., to devise better optimization heuristics and parameterizations for data-driven mirror descent (Tan et al., 2023), to design convolutional networks which are more robust to noise (Sivaprasad et al., 2021), and to enforce related shape-specific properties on level sets (Nesterov et al., 2022).

Third, convexity lies at the heart of classical shape-restricted statistics problems such as convex regression (Seijo & Sen, 2011; Hannah & Dunson, 2013; Guntuboyina & Sen, 2015; Balázs et al., 2015) and log-concave density estimation (Samworth, 2018). Convex energy parameterizations have recently also been used to scale such problems with deep learning infrastructure (Lin & Ba, 2023) and interact naturally with score-based learning objectives (Hyvärinen, 2005; Song et al., 2021), see Feng et al. (2026) for a recent example of shape-constrained score-based regression.

Despite these motivations, the performance of existing convex neural architectures in small to moderate dimensions is often only comparable to that of unconstrained (multi-layer perceptron, MLP)

networks, even when convexity is known to hold, see Section D.2.5 of the supplementary material, in particular for deeper architectures. A potential reason is that most input-convex constructions are not easy to train at depth, and thus fail to leverage the approximation benefits of deeper architectures. Moreover, as we show in Section 3, ICNNs are provably inefficient at approximating quadratic functions, which are fundamental building blocks for convex functions. Our architecture aims to resolve both issues.

Neural networks which are convex in the input. *Input convex neural networks* (ICNNs) (Amos et al., 2017) are the dominant architecture to guarantee convexity with respect to the inputs. A principled initialization strategy improves the practical performance of ICNNs (Hoedt & Klambauer, 2023). Modifications to ICNNs have been proposed to improve their expressiveness, e.g., by using bilinear positive definite skip connections from the input (Bunne et al., 2023).

Based on the representation of convex piecewise-affine functions as maxima of affine forms, alternative input convex network constructions focus on maxout units (Goodfellow et al., 2013). This includes groupwise maximum networks (Warin, 2024) for convex approximation and input convex Kolmogorov–Arnold networks (KANs) (Warin, 2026; Deschatre & Warin, 2026; Thakolkaran et al., 2025). Despite the conceptual appeal of these architectures, they have not been able to consistently outperform ICNNs in practice, and their quantitative approximation properties are not well understood.

Contributions. We introduce *Hyper Input Convex Neural Networks (HyCNNs)*, a novel shape-constrained architecture designed to approximate convex functions while leveraging depth and remaining trainable at scale. At a high level, a HyCNN combines two ideas: (i) parameter constraints as in ICNNs to ensure convexity, and (ii) using maxout units to allow each layer to refine the piecewise-affine structure of the approximation. With appropriate non-negativity constraints on hidden-to-hidden weights, HyCNNs are convex by construction (Proposition 2.1), include standard ReLU ICNNs as a special case, and inherit known universal approximation guarantees for convex functions (Amos et al., 2017; Chen et al., 2019). We also provide a principled initialization strategy for HyCNNs that ensures stable signal propagation across layers at initialization, which is crucial for training deep networks.

Our core technical insight is that the modified convex unit fundamentally changes the *depth–expressiveness trade-off*. Specifically, compared to a classical ICNN layer, each HyCNN layer can *double* the number of affine pieces in the induced piecewise-affine approximation, leading to an exponential-in-depth refinement mechanism. We formalize this by proving that constant-width HyCNNs approximate the quadratic function on $[0, 1]$ with uniform error decaying as $O(2^{-2L})$ in the number of layers L (Theorem 3.3). In contrast, ReLU-based ICNNs in one dimension remain convex and piecewise-affine, but their number of linear pieces grows only linearly with depth (Proposition 3.1), which yields a polynomial lower bound on the best achievable uniform error for $x \mapsto x^2$ (Theorem 3.2). This gap provides a concrete explanation for why "making ICNNs deeper" often fails to deliver commensurate gains in practice.

We then apply HyCNNs to learn high-dimensional OT maps, where we see that they often outperform multiple variants of ICNNs and the Monge gap MLP estimator in terms of predictive performance. In addition, we demonstrate the versatility of HyCNNs by applying them to various synthetic regression problems, demonstrating superior performance over existing convex neural architectures.

Paper organization. Section 2 introduces the HyCNN architecture, establishes convexity, and discusses initialization and training rules. Section 3 compares the approximation properties of HyCNNs and ICNNs. Section 4 presents applications of HyCNNs to optimal transport map estimation. Section 5 provides empirical results. The supplementary material contains additional results to our approximation theory, omitted proofs, and additional experiments and data analysis.

2 HyCNNs: Hyper Input Convex Neural Networks

Architecture. Our architecture of the HyCNN takes inspiration from maxout networks (Goodfellow et al., 2013) and input convex neural networks (Amos et al., 2017). To introduce HyCNNs, we first recall the construction of ICNNs. Formally, an ICNN takes a d-dimensional input $\mathbf{x} \in \mathbb{R}^d$ and outputs a real number; it exhibits $L \in \mathbb{N}$ hidden layers and the number of hidden neurons per layer is characterized by $(d_1, \dots, d_L) \in \mathbb{N}^L$. The output of each layer and the function are given by the

following equations for $\ell = 0, \dots, L-1$ (see Figure 1(a)):

$$\mathrm{ICNN}(\mathbf{x}) := \mathbf{z}_{L+1} := V_L \mathbf{z}_L + W_L \mathbf{x} + \mathbf{b}_L, \qquad \mathbf{z}_{\ell+1} := \sigma(V_\ell \mathbf{z}_\ell + W_\ell \mathbf{x} + \mathbf{b}_\ell), \tag{1}$$

where $\mathbf{z}_0 := \mathbf{0}_d$ and $\mathbf{z}_\ell$ is given by the layerwise activation function $\sigma\colon \mathbb{R} \to \mathbb{R}$ applied componentwise, with parameters consisting of weight matrices and biases. To ensure that $\mathrm{ICNN}(\mathbf{x})$ is convex in $\mathbf{x}$, the activation function σ is convex and non-decreasing (e.g., ReLU), the weight-matrices $\{V_\ell\}_{\ell=0}^{L}$ are constrained to be non-negative, while $\{W_\ell\}_{\ell=0}^{L}$ and $\{\mathbf{b}_\ell\}_{\ell=0}^{L}$ are unconstrained (Amos et al., 2017, Proposition 1).

For the HyCNN architecture, we modify the layerwise construction of ICNNs by introducing an additional lane of matrix operations. Concretely, a HyCNN with L layers and $(d_1, \dots, d_L)$ hidden neurons per layer is given by the following equations for $\ell = 0, \dots, L-1$:

$$\mathrm{HyCNN}(\mathbf{x}) := \mathbf{z}_{L+1} = V_L \mathbf{z}_L + W_L \mathbf{x} + \mathbf{b}_L, \qquad \mathbf{z}_{\ell+1} := \overline{\sigma}\left((V_\ell^k \mathbf{z}_\ell + W_\ell^k \mathbf{x} + \mathbf{b}_\ell^k)_{k\in\{1,2\}}\right), \tag{2}$$

where $\mathbf{z}_0 := \mathbf{0}_d$ and $\overline{\sigma}$ represents the *(gating) activation function* $\overline{\sigma}\colon \mathbb{R}^2 \to \mathbb{R}$ that is convex and componentwise non-decreasing, and the notation $\overline{\sigma}\left((a_k)_{k\in\{1,2\}}\right)$ denotes the vector with entries $\overline{\sigma}(a_1, a_2)$, applied componentwise. The parameters consist of weight matrices and biases with appropriate dimensions to ensure compatibility.

Based on this construction, we have the following convexity guarantee for HyCNNs.

Proposition 2.1. *Let $\overline{\sigma}\colon \mathbb{R}^2 \to \mathbb{R}$ be convex and componentwise non-decreasing, and let the matrices $\{V_\ell^1, V_\ell^2\}_{\ell=0}^{L-1}$ and V_L be componentwise non-negative. Then,* $\mathrm{HyCNN}\colon \mathbb{R}^d \to \mathbb{R}$ *in* (2) *is convex.*

Proof. Note that $\mathbf{z}_0 = \mathbf{0}_d$ is a convex function of $\mathbf{x}$. For the induction step, assume that each component of $\mathbf{z}_\ell$ is a convex function. Then, since every component of V_ℓ^k is non-negative, for each $k \in \{1, 2\}$ each component of $V_\ell^k \mathbf{z}_\ell + W_\ell^k \mathbf{x} + \mathbf{b}_\ell^k$ is convex. Since $\overline{\sigma}$ is convex and componentwise non-decreasing, $\mathbf{z}_{\ell+1}$ is convex as well (see Lemma B.1). Finally, the output layer is given by a non-negative affine combination of convex functions plus an affine function, which is again a convex function. Thus, by induction, HyCNN is convex. □

By choosing $\overline{\sigma}(a_1, a_2) = \sigma(a_1)$, where $\sigma\colon \mathbb{R} \to \mathbb{R}$ is a convex and non-decreasing activation function such as the ReLU, we recover the standard ICNN architecture. Hence, HyCNNs can be interpreted as a generalization of ICNNs, and enjoy similar universal approximation guarantees for convex functions as classical ICNNs (Chen et al., 2019).

In this paper, we consider activation functions of the form

$$\overline{\sigma}_\tau(a_1, a_2) = \begin{cases} \tau \log\left(e^{a_1/\tau} + e^{a_2/\tau}\right), & \text{if } \tau > 0, \\ \max(a_1, a_2), & \text{if } \tau = 0. \end{cases} \tag{3}$$

The componentwise maximum $\overline{\sigma}_0$ is also known as a *maxout unit* (Goodfellow et al., 2013) and yields piecewise-affine HyCNNs. For $\tau > 0$, the log-sum-exp function $\overline{\sigma}_\tau$ is a smooth approximation of the componentwise maximum. Lemma B.3 and the surrounding discussion show that for activation function $\overline{\sigma}_\tau$ with $\tau \geq 0$, the entrywise non-negativity of the hidden-to-hidden weight matrices $\{V_\ell^1, V_\ell^2\}_{\ell=0}^{L-1}$ is necessary to ensure that all HyCNNs are input convex functions.

Remark 2.2 (Comparison with Maxout and GroupMax networks). HyCNNs exhibit some similarities and differences with Maxout networks (Goodfellow et al., 2013) and GroupMax networks (Warin, 2024). Unlike HyCNNs, in Maxout networks the weights for hidden-to-hidden neuron connections are not constrained to be non-negative. Imposing this constraint results in the GroupMax network architecture. A key distinction of HyCNNs to both Maxout and GroupMax networks is the presence of (distinct) skip connections entering into both inputs of the maxout unit. This is a central technical device in our approximation theory of the quadratic function, and seems to stabilize the gradient norm of parameters across different layers. Our empirical results in Section D.2 also suggest that this architectural change is crucial for training deeper networks.

Initialization of HyCNNs. To train HyCNNs, in particular, deep architectures, the gradient norms of parameters across different layers need to be stable at initialization to mitigate the vanishing/exploding gradient problem and to ensure that all layers are effectively trained. In the following, we introduce a suitable initialization scheme, see Figure 2 for numerical evidence, which ensures that the signal

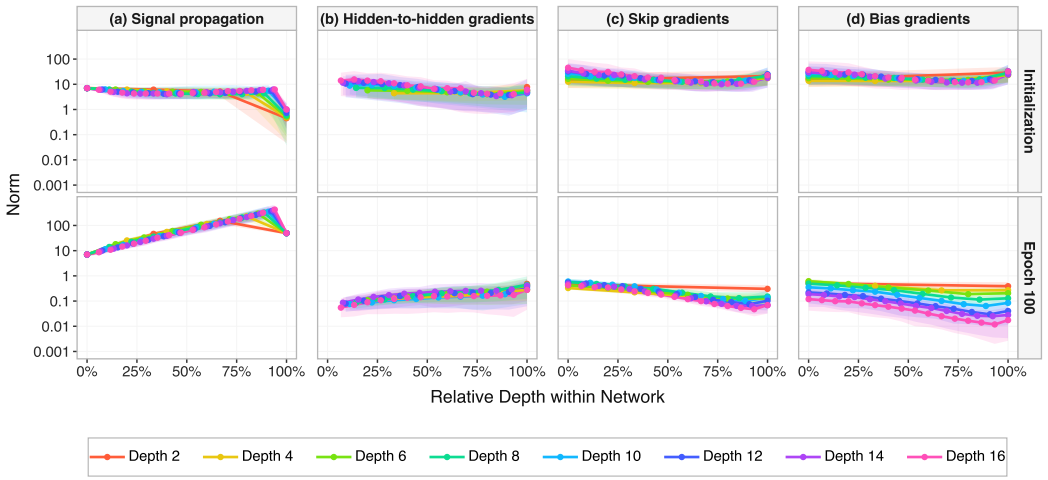

Figure 2: Signal propagation and gradient profiles across relative depth in HyCNN at initialization (top row) and after 100 epochs of training (bottom row) for width $W = 40$, depths $L \in \{2, 4, \ldots, 16\}$ (colors), and input dimension $d = 50$. The relative depth within the network refers to the respective layer index normalized by network depth, with 0% referring to the input, and 100% to the scalar output of the network. **(a)** Signals are computed from a single forward pass at initialization with standard Gaussian input $\mathbf{X} \sim \mathcal{N}(0, I_d)$. Each curve is the mean hidden-state ℓ_2-norm of the neurons over 100 random seeds. The shaded bands show the 10th–90th percentile range. 0% corresponds to the input norm $\|\mathbf{X}\|_2$ (which is close to $\sqrt{d} \approx 7.07$). **(b-d)** Gradients are computed from a single backward pass at initialization for least-squares regression on synthetic data of size $n = 5{,}000$, with $\mathbf{X}_i \sim \mathcal{N}(0, I_d)$ and $Y_i = \|\mathbf{X}_i\|_2^2 + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 1)$. Batch size for training equals 1,000. Each curve is the mean of the gradient norm for (b) hidden-to-hidden weight,[2] (c) skip weights, and (d) bias weights. The shaded bands show the 10th–90th percentile range.

propagation and gradient profiles are well-behaved across layers at initialization, and lead to stable training dynamics of most weights across the network, with a tendency for gradients of deeper layers to be slightly smaller than those of shallower layers. After 100 epochs of training, the network parameters still exhibit stable gradient norms, while the signal norm across layers appears to grow.

Due to the non-negativity constraints of the hidden-to-hidden weights, standard initialization strategies based on centered weight distributions, which are commonly adopted in unconstrained neural networks (LeCun et al., 1998; He et al., 2015), are not directly applicable to HyCNNs. Inspired by the approach of Hoedt & Klambauer (2023) for ICNNs, we develop in the following an initialization strategy for HyCNNs. The key objective of our approach is to ensure that pre-activations exhibit comparable statistical properties across all layers at initialization. To this end, we examine how the statistics of the pre-activations evolve from one layer to the next and derive fixed-point equations that keep their mean, variance, and covariance stable.

More specifically, for $\ell \in [L-1]$ let

$$\mathbf{s}_{\ell+1} = (s_{\ell+1,i})_{i\in[d_{\ell+1}]} := V_\ell^1 \mathbf{z}_\ell + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1, \qquad \mathbf{t}_{\ell+1} = (t_{\ell+1,i})_{i\in[d_{\ell+1}]} := V_\ell^2 \mathbf{z}_\ell + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2.$$

Notice that $\mathbf{z}_\ell = \max(\mathbf{s}_\ell, \mathbf{t}_\ell)$. At initialization, the entries of V_ℓ^1 and V_ℓ^2 are drawn i.i.d. from a distribution supported on $\mathbb{R}_+$. The entries of W_ℓ^1 and W_ℓ^2 are drawn i.i.d. from a centered Gaussian distribution, while the entries of $\mathbf{b}_\ell^1$ and $\mathbf{b}_\ell^2$ are set to a fixed value. Since the variables $s_{\ell,1}, t_{\ell,1}, \ldots, s_{\ell,d_\ell}, t_{\ell,d_\ell}$ are exchangeable as a collection of $2d_\ell$ variables,[3] it is sufficient to impose the following fixed-point equations

$$\mathbb{E}(s_{\ell+1,1}) = \mathbb{E}(s_{\ell,1}), \qquad \mathbb{E}(s_{\ell+1,1}^2) = \mathbb{E}(s_{\ell,1}^2), \qquad \mathbb{E}(s_{\ell+1,1} t_{\ell+1,1}) = \mathbb{E}(s_{\ell,1} t_{\ell,1})$$

to ensure that the mean, variance, and covariance of all neuron pre-activations are stable across layers at initialization and thus induce a well-conditioned signal propagation. To obtain explicit solutions to

[2] We reparameterize the hidden-to-hidden weights as $V_\ell = \mathrm{softplus}(R_\ell)$ for unconstrained matrices R_ℓ and softplus applied element-wise. The gradient in the figure depicts the gradient norm of R_ℓ across different layers.

[3] A collection of random variables $x_1, \ldots, x_n$ is exchangeable if its joint distribution is invariant under permutations.

these equations, we assume that $\|\mathbf{x}\|_2^2 \approx d$ and use a Gaussian approximation of the joint distribution of $s_{\ell,1}, \dots, s_{\ell,d_\ell}, t_{\ell,1}, \dots, t_{\ell,d_\ell}$.

Initialization scheme for HyCNNs:

- The entries of V_ℓ^1 and V_ℓ^2 for $\ell \in [L-1]$ are drawn i.i.d. from a distribution supported on $\mathbb{R}_+$ with mean and variance, respectively, given by

$$\mu := \left(d_\ell^2 + (1 - \tfrac{1}{\pi})d_\ell\right)^{-1/2}, \quad \sigma^2 := \frac{1}{4d_\ell}.$$

  We take the log-normal distribution as it is easy to sample from. The above constraints are then realized by drawing

$$X := \exp(\widetilde{X}), \quad \widetilde{X} \sim \mathcal{N}\left(\ln\left(\frac{\mu^2}{\sqrt{\tau}}\right), \ln\left(\frac{\tau}{\mu^2}\right)\right) \quad \text{with } \tau := \mu^2 + \sigma^2,$$

  where $\mathcal{N}(\mu, \sigma^2)$ denotes the normal distribution with mean μ and variance σ^2.
- The entries of W_ℓ^1 and W_ℓ^2 for $\ell \in [L-1]$ are drawn i.i.d. from $\mathcal{N}(0, 1/(4d))$.
- The entries of $\mathbf{b}_\ell^1$ and $\mathbf{b}_\ell^2$ for $\ell \in [L-1]$ are set to $-\sqrt{\frac{d_\ell}{2\pi d_\ell + 2\pi - 2}}$.
- The parameters $V_L, W_L, \mathbf{b}_L$ in the last layer use the same initialization scheme.
- The parameters W_0^1, W_0^2 and $\mathbf{b}_0^1, \mathbf{b}_0^2$ in the first layer adopt the initialization of LeCun et al. (1998) (i.e., entries are i.i.d. samples from $N(0, 1/d)$).

Derivations are provided in Appendix C. We note that the initialization scheme for HyCNNs is motivated by that of ICNNs proposed by Hoedt & Klambauer (2023), but tailored to the architectural differences of HyCNNs. Based on this initialization scheme, we empirically verify that the signals across different layers are stable at initialization (see Figure 2(a)), which is crucial for training deep networks. It is surprising that there is such a close agreement across the relative depth profiles. It suggests that a neural ODE limit (Cohen et al., 2021) might exist for HyCNNs. For a visualization of HyCNNs at initialization across different depths see Figure 8.

3 Approximation of Quadratic Functions

The quadratic function $x \mapsto x^2$ is fundamental for approximating more complex convex functions, and thus serves as a natural testbed for understanding the approximation capabilities of convex neural architectures. In the following we analyze the capabilities of ICNNs and HyCNNs in approximating the quadratic function on the unit interval $[0, 1]$. While the focus of this section is on the univariate setting for simplicity, the results can be easily extended to the multivariate setting, and we discuss this in Appendix A. All proofs of this section are deferred to Appendix B.

Limited Expressivity of ICNNs. A preliminary observation is that (standard or leaky) ReLU ICNNs are piecewise-affine, and the number of linear pieces grows only linearly with the number of layers. Recall that a leaky ReLU is defined as $\sigma(x) = \max(\alpha x, x)$ for some $\alpha \in (0, 1)$.

Proposition 3.1. *Consider an ICNN $f : \mathbb{R} \to \mathbb{R}$ with L hidden layers and d_ℓ neurons on layer ℓ, whose activation function is given by a (standard or leaky) ReLU. Then, f is piecewise-affine with at most $d_1 + 2d_2 + \cdots + 2d_L$ pieces.*

Theorem 3.2. *Let $\mathcal{G}_k$ be the set of real-valued, piecewise-affine, convex functions on $[0, 1]$ with at most k pieces. Then,*

$$\inf_{g \in \mathcal{G}_k} \int_0^1 \left|g(x) - x^2\right| dx \geq \frac{1}{16k^2}, \quad \text{and} \quad \inf_{g \in \mathcal{G}_k} \sup_{x \in [0,1]} |g(x) - x^2| \geq \frac{1}{8k^2}. \tag{4}$$

In particular, any (standard or leaky) ReLU ICNN $f : \mathbb{R} \to \mathbb{R}$ with L hidden layers and d_ℓ neurons in layer ℓ satisfies

$$\int_0^1 \left|f(x) - x^2\right| dx \geq \frac{1}{16(d_1 + 2\sum_{\ell=2}^L d_\ell)^2} \quad \text{and} \quad \sup_{x \in [0,1]} |f(x) - x^2| \geq \frac{1}{8(d_1 + 2\sum_{\ell=2}^L d_\ell)^2}.$$

Expressivity of HyCNNs. Next, we focus on the approximation capabilities of HyCNNs.

Theorem 3.3 (Quadratic Function Approximation)**.** *For any $L \in \mathbb{N}$, even numbers $d_1, \dots, d_L$ and $\tau \geq 0$, there exists a HyCNN $h\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$, L hidden layers, and $(d_1, \dots, d_L)$ neurons per layer such that*

$$\sup_{x \in [0,1]} |h(x) - x^2| \leq \frac{1}{7.99 \prod_{\ell=1}^{L} d_\ell^2}.$$

A first key takeaway from Theorem 3.3 is that wider HyCNNs lead to a *polynomial* increase of the approximation accuracy, while deeper networks lead to an *exponential* increase of the approximation accuracy. In particular, HyCNNs are able to *leverage depth* for approximation, which is in stark contrast to ICNNs. Our empirical findings show that deeper HyCNNs can also be effectively trained.

Remark 3.4 (Interpretation of Theorem 3.3). The proof is detailed in Appendix B. A similar result is well-known for feedforward ReLU networks. However, the known construction crucially relies on highly oscillating sawtooth functions, which are not input convex (Telgarsky, 2016; Yarotsky, 2017; Schmidt-Hieber, 2020). It is therefore surprising that each HyCNN with at least two neurons per layer can at least *double* the number of linear pieces in the piecewise-affine approximation of x^2, leading to an exponential decay of the approximation error with respect to the number of layers. To explain why it works, we derive a second proof for the case $d_1 = \dots = d_L = 2$ that reveals how the oscillatory functions can be cached in input convex functions.

Based on our construction for the quadratic function, we also obtain an efficient approximation of higher order monomials $x \mapsto x^n$ on $[0, 1]$ for $n = 2, 3, \dots$ by suitably concatenating HyCNN blocks.

Theorem 3.5 (Monomial Function Approximation)**.** *For any integer $n \geq 2$, $L \in \mathbb{N}$ and odd number $m \geq 3$ and $\tau \geq 0$, there exists a HyCNN $h\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$, $\lceil \log_2(n) \rceil L$ layers and m neurons per layer such that*

$$\sup_{x \in [0,1]} \left| h(x) - x^n \right| \leq \frac{n}{1.99} (m-1)^{-2L}.$$

The underlying HyCNN construction is not a straightforward extension of the network construction for the quadratic function (Theorem 3.3), and requires a more involved proof strategy. This is in contrast to feedforward ReLU networks where the approximation of, say, $x \mapsto x^3$ first constructs a network approximately computing $x \mapsto (x, x^2)$ and then concatenates the outcome with a multiplication network that approximates the map $(u, v) \mapsto uv$, therefore computing approximately $x \mapsto xx^2 = x^3$. However, $(u, v) \mapsto uv$ is not a convex function and therefore not representable by a HyCNN.

The exponential approximation rates for monomials can also be used to derive approximation bounds for analytic functions whose Taylor coefficients of order two and higher are non-negative. In addition, the approximation advantage of HyCNNs over standard ICNNs extends to gradient approximation when the target function is convex and smooth, which is useful for applications in optimal transport. See Theorem A.3 and Proposition A.4 in Appendix A for formal statements.

We close this section with further implications of the above results. Denote by $\mathrm{ICNN}_d(L, m)$, $\mathrm{HyCNN}_d(L, m)$ and $\mathrm{ReLU}_d(L, m)$ the respective classes of all (ReLU) ICNNs, (maxout) HyCNNs and ReLU networks with d input neurons, L hidden layers and m neurons per layer, respectively. As a direct consequence of Theorems 3.2 and 3.3 we have that whenever m' is even and $(m')^{L'} \geq 2mL$,

$$\mathrm{HyCNN}_d(L', m') \subsetneq \mathrm{ICNN}_d(L, m) \subseteq \mathrm{HyCNN}_d(L, m) \subseteq \mathrm{ReLU}_d(L, 3m + 2d). \tag{5}$$

This shows that a much larger number of ICNN parameters is needed to represent all HyCNN functions. The $\subsetneq$ part also reveals that ICNNs are limited in their ability to represent the maximum of two functions as otherwise they could emulate HyCNNs contradicting (5). The last inclusion implies that any lower bound on the approximation error of ReLU networks directly translates to those of HyCNNs. Moreover, known bounds on the VC dimension and covering number of ReLU networks imply corresponding bounds for HyCNNs. A formal proof of (5) is given in Appendix B.5.

4 Application: Neural Optimal Transport Map Learning

Learning an optimal transport (OT) map between probability measures is a central problem with broad impact in machine learning (Courty et al., 2017; del Barrio et al., 2019), statistics (Hallin

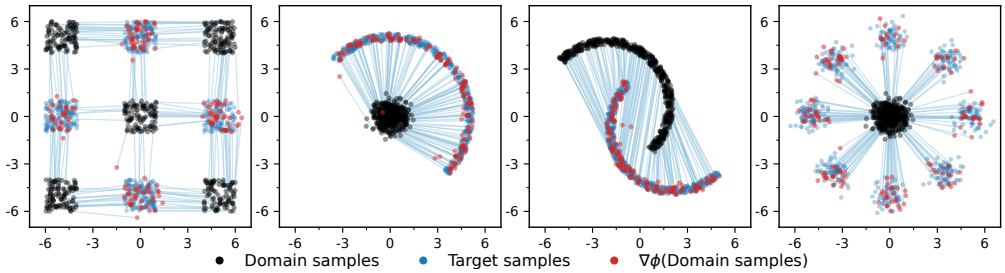

Figure 3: Pushforwards under learned HyCNN OT map (width 40, depth 4). The HyCNN is trained for $n = m = M = 2{,}000$ data points for 2500 outer iterations for learning ϕ, with $S_{\text{in}} = 10$ steps for the inner loop, via Adam and decaying learning rates and smoothing parameter τ.

et al., 2021; De Lara et al., 2024), and natural sciences (Wang et al., 2011; Schiebinger et al., 2019; Tameling et al., 2021; Bunne et al., 2023). To formalize the problem, consider probability measures P and Q on $\mathbb{R}^d$ with finite second moments. For squared Euclidean costs, the *OT map* is defined as

$$\overline{T} \in \operatorname*{argmin}_{T:\, T_{\#}P=Q} \int_{\mathbb{R}^d} \|\mathbf{x} - T(\mathbf{x})\|^2 \, dP(\mathbf{x}), \qquad \text{where } T_{\#}P := P \circ T^{-1},$$

assuming it exists. If P and Q are absolutely continuous, Brenier's theorem (Santambrogio, 2015, Theorem 1.22) guarantees the existence of a P-a.s. unique convex potential $\phi : \mathbb{R}^d \to \mathbb{R}$, the *OT potential*, such that $\overline{T} = \nabla\phi$. This links OT map estimation with learning the OT potential.

Recent work has produced a variety of estimators for $\overline{T}$, including plug-in methods (Manole et al., 2024), entropic approaches (Seguy et al., 2018; Pooladian & Niles-Weed, 2021), semidual formulations (Hütter & Rigollet, 2021; Divol et al., 2025), and neural estimators (Taghvaei & Jalali, 2019; Makkuva et al., 2020; Bunne et al., 2023; Uscidda & Cuturi, 2023). It is worth noting that empirical plug-in and entropic estimators, though adaptive to low-dimensional structures (Weed & Bach, 2019; Hundrieser et al., 2024; Groppe & Hundrieser, 2024; Stromme, 2024; Divol et al., 2025), are oftentimes outperformed by neural estimators in genuinely high-dimensional settings (Makkuva et al., 2020; Uscidda & Cuturi, 2023), which motivates the use of neural OT in this regime.

We use the variational formulation of Makkuva et al. (2020), based on the semidual framework by Hütter & Rigollet (2021), which characterizes the OT potential ϕ via a saddle-point problem,

$$\phi \in \operatorname*{argmax}_{\text{convex } f \in L^1(P),\, f^* \in L^1(Q)} \; \inf_{g \in L^1(Q)} \; -\mathbb{E}_{\mathbf{X}\sim P}[f(\mathbf{X})] - \mathbb{E}_{\mathbf{Y}\sim Q}\big[\langle \mathbf{Y}, \nabla g(\mathbf{Y})\rangle - f(\nabla g(\mathbf{Y}))\big].$$

Given f, the optimal g corresponds to the convex conjugate of f, i.e., $g = f^*$. In practice, we replace the expectations by empirical averages (using mini-batches), and the function classes are restricted to neural network families. Denoting the training data $(\mathbf{x}_i)_{i=1}^n \sim P$ and $(\mathbf{y}_j)_{j=1}^m \sim Q$, we follow the mini-batch training protocol of Makkuva et al. (2020) but replace the ICNNs by HyCNNs with activation function $\overline{\sigma}_\tau(a_1, a_2) = \tau \log(e^{a_1/\tau} + e^{a_2/\tau})$. This leads to a saddle-point problem for mini-batches $(\mathbf{X}_i)_{i=1}^M$ and $(\mathbf{Y}_j)_{j=1}^M$ from $(\mathbf{x}_i)_{i=1}^n$ and $(\mathbf{y}_j)_{j=1}^m$, respectively, of size $M \in \mathbb{N}$,

$$\widehat{\phi} \in \operatorname*{argmax}_{f \in \texttt{HyCNN}_\tau(\mathbb{R}^d)} \; \min_{g \in \texttt{HyCNN}_\tau(\mathbb{R}^d)} \; -\tfrac{1}{M}\left(\textstyle\sum_{i=1}^M f(\mathbf{X}_i) + \langle \mathbf{Y}_i, \nabla g(\mathbf{Y}_i)\rangle - f(\nabla g(\mathbf{Y}_i))\right). \tag{6}$$

The *HyCNN-based OT map estimator* is then defined by $\widehat{T} := \nabla\widehat{\phi}$, see Figure 3 for an illustration. The training method is detailed in Algorithm 1. In Section D.3, it uses Adam (Kingma & Ba, 2015) and involves an outer loop (with S_{out} iterations) and an inner loop (with S_{in} iterations per outer iteration). During training we parameterize hidden-to-hidden weights via a softplus function to ensure non-negativity, deviating from the approach by Makkuva et al. (2020), who use a regularization term for a soft-enforcement of non-negativity.

The simulations in Appendix D.3, make two key observations on the choice of τ. In small-dimensional settings, a small $\tau > 0$ better captures the discontinuities in the OT map. A cosine annealing schedule for τ often leads to improved performance compared to using a fixed τ. In moderate to high-dimensional settings, a positive fixed τ during training and chosen according to a Sinkhorn divergence

evaluation on a validation set tends to work well. Given that HyCNNs empirically appear to already often perform well for relatively short training, i.e., S_{out} in the range of 1,000 to 5,000 iterations and S_{in} in the range of 5 to 10 steps per outer iteration, which can be realized in a matter of minutes, it is feasible to tune τ by some validation criterion without incurring significant computational overhead.

5 Experiments

Convex regression. We draw covariates $(\mathbf{X}_i)_{i=1}^n$ from $\text{Unif}[-1,1]^d$ with responses according to the model $Y_i = f(\mathbf{X}_i) + \epsilon_i$, where $\epsilon_i \sim N(0,\sigma^2)$ are independent Gaussian errors with standard deviation $\sigma \geq 0$, and $f\colon \mathbb{R}^d \to \mathbb{R}$ is one of the following functions,

$$\text{(a) } f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2, \qquad \text{(b) } f_2(\mathbf{x}) = \|\mathbf{x}\|_4^4, \qquad \text{(c) } f_3(\mathbf{x}) = \|\mathbf{x}\|_2^2 + 0.25\sin\left(20\|\mathbf{x}\|_2\right)$$

Note that f_3 is not convex, but it can be fairly well approximated by a convex function. We consider dimension $d = 50$, training sample sizes $n \in \{100, 200, 500, 1{,}000, 2{,}000, 5{,}000, 1{,}0000\}$, and Gaussian noise with variance $\sigma^2 = 1$ for f_1, f_2 and $\sigma = 0$ for f_3. For each regression setting, we train feedforward ReLU networks (multilayer perceptrons, MLPs), ICNNs (Amos et al., 2017), GroupMax networks (GMNs, Warin 2024, i.e., HyCNNs without skip connections from the input), and HyCNNs. We use widths $m = 84$ for MLPs, $m = 64$ for ICNNs, $m = 60$ for GMNs, and $m = 40$ for HyCNNs, yielding similar parameter counts in dimension $d = 50$, see Table 1. Further we consider depth $L \in \{2, 4, 8, 16, 32\}$ for all architectures. Section D.2.1 contains additional details.

In Figure 4, we show the prediction MSE on an independent test set of size 1,000. For clarity, we only display results for HyCNNs with depths $L \in \{2, 4, 8, 16\}$ and the method which performs best for training sample size $n = 10{,}000$. HyCNNs with depth at least four consistently outperform all other methods across all regression settings, with deeper HyCNNs performing better than shallower ones. The Appendices D.2.2–D.2.5, cover extensive additional experimental results showing that HyCNNs consistently outperform the baseline methods. We refer to Appendix D.2.6 for an analysis of the effect of the smoothing parameter on signal propagation and regression performance, and to Appendix D.2.7 for an empirical evaluation of the effect of the number of activation branches. Overall, deep HyCNNs with small τ and two branches achieve the best performance.

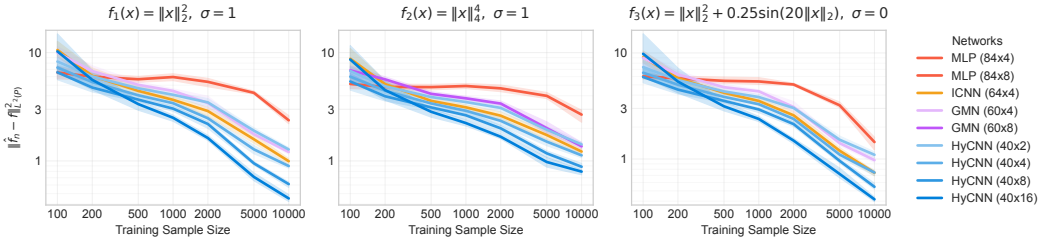

Figure 4: Prediction MSE for different training sample sizes n for the 50-dimensional regression tasks with 10th-90th percentile bands across 10 runs. Each panel shows HyCNNs with depths $2, 4, 8, 16$ and the best performing MLP, ICNN, GMN at training sample size $n = 10^4$ among depths $\in \{2, 4, 8, 16\}$.

Neural OT map estimation. To assess the performance of the HyCNN-based OT map estimator, we consider the following OT potentials with corresponding OT maps given by their gradients,

$$\text{(a) } \phi_1(\mathbf{x}) = \frac{\|\mathbf{x}\|_2^2}{2}, \quad T_1(\mathbf{x}) = \mathbf{x}, \quad \text{(b) } \phi_2(\mathbf{x}) = \tfrac{1}{2}\textstyle\sum_{i=1}^d \left(1 + \frac{\sin(i)}{2}\right)x_i^2, \quad T_2(\mathbf{x}) = \left((1 + \frac{\sin(i)}{2})x_i\right)_{i=1}^d,$$

and draw $n = m = 5{,}000$ independent samples from $P = N(0, I_d)$ and $Q = (\nabla\phi_\ell)_{\#}P$ for $\ell \in \{1, 2\}$ with $d = 50$. In Figure 5**(a)** we display the (dimension-normalized) prediction MSE for an independent testing set of size 1,000 for the two displayed OT map estimation tasks for $d = 50$ across training iterations. We compare HyCNNs of width 44, depth 4, and smoothness $\tau = 10$ with ICNN variants (Makkuva et al., 2020) of width 64 and baseline depths $L \in \{2, 4, 6\}$. The Monge gap MLP estimator (Uscidda & Cuturi, 2023) uses width 104 and the same depths, see Table 13 for a summary of the total parameter counts. HyCNNs outperform all other methods across all tasks, with ICNNs performing better than the Monge gap MLP estimator. Appendices D.3.1 and D.3.2 contain additional

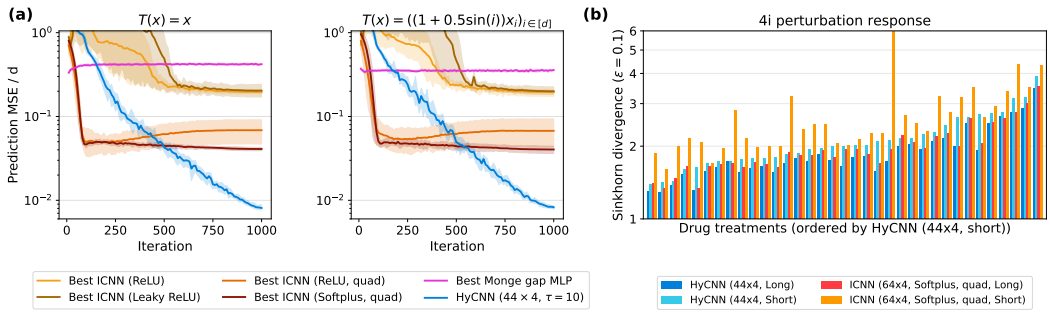

Figure 5: Performance of OT map estimation for synthetic and 4i single cell data. **(a)** Prediction MSE (normalized by dimension $d = 50$) for $n = 5{,}000$ training samples for OT map estimation tasks with 10th-90th percentile bands across 10 runs. Both panels show the same HyCNN (width 44) and the best ICNN (width 64) with ReLU or leaky ReLU, or with a quadratic term in the first layer with ReLU or Softplus activation, and the Monge gap MLP (width 104) estimator, each within a network class of depths $L \in \{2, 4, 6\}$ and $\tau \in \{0.1, 1, 10\}$ whenever softplus or log-sum-exp is used. **(b)** Sinkhorn divergence between predicted and ground-truth drug-perturbed distributions on a 20% held-out test partition of the 4i dataset for each drug based on ICNN (64×4, Softplus, $\tau = 1$, with a quadratic term in the first layer) and HyCNN (44×4, log-sum-exp, $\tau = 1$) trained for $S_{\text{out}} = 1{,}000$ outer iterations and $S_{\text{in}} = 5$ inner iterations (short schedule) as well as $S_{\text{out}} = 50{,}000$ iterations and $S_{\text{in}} = 10$ inner iterations (long schedule).

information on training rules and numerical evidence for two-dimensional and higher-dimensional OT map estimation tasks, respectively. Appendix D.3.3 contains ablation studies for the OT map estimation experiments and reveals that convexity, the two-branch architecture, smoothness, and the initialization all contribute jointly to the performance of the HyCNN-based OT map estimator.

Moreover, for the 4i single-cell dataset, preprocessed by Bunne et al. (2023), we train HyCNNs and ICNNs to map the unperturbed control distribution onto each drug-perturbed distribution and find that HyCNNs attain a lower Sinkhorn divergence on a 20% held-out test partition than the best ICNN baseline across nearly all considered perturbations, see Figure 5**(b)** or Figure 17 for an enlarged view. Details on training rules and further aspects on the data analysis are given in Appendix E. To make the comparison robust to variability across checkpoints, each reported value averages the test-set Sinkhorn divergence over the $K = 10$ checkpoints with smallest validation loss; see Appendix E for details. Figure 5**(b)** contrasts a short and a long training schedule per architecture. Longer training improves performance, but architecture changes outweigh the training budget: with 100 times less compute, the short-trained HyCNN is competitive with the long-trained ICNN, and is consistently outperformed by the long-trained HyCNN.

6 Discussion and Limitations

HyCNNs appear as a promising architecture for learning convex functions in high dimensions: for regression, deeper models perform better than shallower ones, and for OT map estimation, HyCNNs with large τ perform better than those with small τ. We also note that the HyCNN two-lane architecture could also be applied in combination with other frameworks. For example, Convolutional HyCNNs can be defined by constraining the hidden-to-hidden convolution kernels to be non-negative, leaving the input-skip convolutions unconstrained, and applying the two-lane maxout or log-sum-exp gate channelwise. Input-convex convolutional neural networks (Korotin et al., 2021; Amos, 2023) are recovered as the single-lane ($\overline{\sigma}(a_1, a_2) = \sigma(a_1)$) special case.

We note two limitations of our work. By construction, HyCNNs exhibit an inductive bias toward smooth, coercive convex functions, which benefits many applications but may constrain training in settings that deviate from these assumptions. Further, our empirical study is bounded by computational constraints: though we conducted extensive simulations, our training tasks were limited, and a more large-scale study remains a future step.

Acknowledgments and Disclosure of Funding

S. Hundrieser is funded from the German National Academy of Sciences Leopoldina under grant number LPDS 2024-11. I. Kong and J. Schmidt-Hieber are supported by ERC grant A2B (grant agreement number 101124751). S. Hundrieser further acknowledges helpful discussions with Paul Hagemann in an early phase of this project. Claude Code Opus 4.6 and ChatGPT Codex 5.4 and later 6.1 Sol were used for coding, in particular for implementing the neural network architectures.

References

Amos, B. (2023). On amortizing convex conjugates for optimal transport. In *The Eleventh International Conference on Learning Representations*.

Amos, B., Xu, L., & Kolter, J. Z. (2017). Input convex neural networks. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155.: PMLR.

Balázs, G., György, A., & Szepesvári, C. (2015). Near-optimal max-affine estimators for convex regression. In G. Lebanon & S. V. N. Vishwanathan (Eds.), *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 56–64.: PMLR.

Brenier, Y. (1991). Polar factorization and monotone rearrangement of vector-valued functions. *Communications on Pure and Applied Mathematics*, 44(4), 375–417.

Bunne, C., Stark, S. G., Gut, G., del Castillo, J. S., Levesque, M., Lehmann, K.-V., Pelkmans, L., Krause, A., & Rätsch, G. (2023). Learning single-cell perturbation responses using neural optimal transport. *Nature Methods*, 20(11), 1759–1768.

Chen, Y., Shi, Y., & Zhang, B. (2019). Optimal control via neural networks: A convex approach. In *International Conference on Learning Representations*.

Cohen, A.-S., Cont, R., Rossier, A., & Xu, R. (2021). Scaling properties of deep residual networks. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2039–2048.: PMLR.

Courty, N., Flamary, R., Tuia, D., & Rakotomamonjy, A. (2017). Optimal transport for domain adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9), 1853–1865.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems*, volume 26: Curran Associates, Inc.

Cuturi, M., Meng-Papaxanthos, L., Tian, Y., Bunne, C., Davis, G., & Teboul, O. (2022). Optimal transport tools (OTT): A JAX toolbox for all things Wasserstein. *Preprint arXiv:2201.12324*.

De Lara, L., González-Sanz, A., Asher, N., Risser, L., & Loubes, J.-M. (2024). Transport-based counterfactual models. *Journal of Machine Learning Research*, 25(136), 1–59.

del Barrio, E., Gamboa, F., Gordaliza, P., & Loubes, J.-M. (2019). Obtaining fairness using optimal transport theory. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2357–2365.: PMLR.

Deschatre, T. & Warin, X. (2026). Input convex Kolmogorov–Arnold networks. *SIAM Journal on Scientific Computing*, 48(3), C579–C603.

Divol, V., Niles-Weed, J., & Pooladian, A.-A. (2025). Optimal transport map estimation in general function spaces. *The Annals of Statistics*, 53(3), 963–988.

Eckle, K. & Schmidt-Hieber, J. (2019). A comparison of deep networks with ReLU activation function and linear spline-type methods. *Neural Networks*, 110, 232–242.

Feng, O. Y., Kao, Y.-C., Xu, M., & Samworth, R. J. (2026). Optimal convex M-estimation via score matching. *The Annals of Statistics*, 54(1), 408–441.

Feydy, J., Séjourné, T., Vialard, F.-X., Amari, S.-i., Trouvé, A., & Peyré, G. (2019). Interpolating between optimal transport and MMD using Sinkhorn divergences. In K. Chaudhuri & M. Sugiyama (Eds.), *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 2681–2690.: PMLR.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28(3) of *Proceedings of Machine Learning Research*, pages 1319–1327.: PMLR.

Groppe, M. & Hundrieser, S. (2024). Lower complexity adaptation for empirical entropic optimal transport. *Journal of Machine Learning Research*, 25(344), 1–55.

Guntuboyina, A. & Sen, B. (2015). Global risk bounds and adaptation in univariate convex regression. *Probability Theory and Related Fields*, 163(1–2), 379–411.

Hallin, M., del Barrio, E., Cuesta-Albertos, J., & Matrán, C. (2021). Distribution and quantile functions, ranks and signs in dimension d: A measure transportation approach. *The Annals of Statistics*, 49(2), 1139–1165.

Hannah, L. A. & Dunson, D. B. (2013). Multivariate convex regression with adaptive partitioning. *Journal of Machine Learning Research*, 14(102), 3261–3294.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

Hoedt, P.-J. & Klambauer, G. (2023). Principled weight initialisation for input-convex neural networks. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in Neural Information Processing Systems*, volume 36, pages 46093–46104.: Curran Associates, Inc.

Huang, C.-W., Chen, R. T. Q., Tsirigotis, C., & Courville, A. (2021). Convex potential flows: Universal probability distributions with optimal transport and convex optimization. In *International Conference on Learning Representations*.

Hundrieser, S., Staudt, T., & Munk, A. (2024). Empirical optimal transport between different measures adapts to lower complexity. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 60(2), 824–846.

Hütter, J.-C. & Rigollet, P. (2021). Minimax estimation of smooth optimal transport maps. *The Annals of Statistics*, 49(2), 1166–1194.

Hyvärinen, A. (2005). Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24), 695–709.

Kingma, D. P. & Ba, J. L. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *International Conference on Learning Representations*.

Korotin, A., Egiazarian, V., Asadulaev, A., Safin, A., & Burnaev, E. (2021). Wasserstein-2 generative networks. In *International Conference on Learning Representations*.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient BackProp. In G. B. Orr & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer.

Liang, S. & Srikant, R. (2017). Why deep neural networks for function approximation? In *International Conference on Learning Representations*.

Lin, A. & Ba, D. (2023). How to train your FALCON: Learning log-concave densities with energy-based neural networks. In *Fifth Symposium on Advances in Approximate Bayesian Inference*, pages 1–14.

Makkuva, A. V., Taghvaei, A., Lee, J. D., & Oh, S. (2020). Optimal transport mapping via input convex neural networks. In H. Daumé, III & A. Singh (Eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6672–6681.: PMLR.

Manole, T., Balakrishnan, S., Niles-Weed, J., & Wasserman, L. (2024). Plugin estimation of smooth optimal transport maps. *The Annals of Statistics*, 52(3), 966–998.

McClure, D. E. (1975). Nonlinear segmented function approximation and analysis of line patterns. *Quarterly of Applied Mathematics*, 33(1), 1–37.

Nesterov, V., Arend Torres, F., Nagy-Huber, M., Samarin, M., & Roth, V. (2022). Learning invariances with generalised input-convex neural networks. *Preprint arXiv:2204.07009*.

Peyré, G. & Cuturi, M. (2019). Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5–6), 355–607.

Pooladian, A.-A. & Niles-Weed, J. (2021). Entropic estimation of optimal transport maps. In *NeurIPS 2021 Workshop on Optimal Transport and Machine Learning (OTML)*.

Rezende, D. J. & Mohamed, S. (2015). Variational inference with normalizing flows. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538.: PMLR.

Samworth, R. J. (2018). Recent progress in log-concave density estimation. *Statistical Science*, 33(4), 493–509.

Santambrogio, F. (2015). *Optimal transport for applied mathematicians: Calculus of variations, PDEs, and modeling*, volume 87 of *Progress in Nonlinear Differential Equations and Their Applications*. Cham: Birkhäuser.

Schiebinger, G., Shu, J., Tabaka, M., Cleary, B., Subramanian, V., Solomon, A., Gould, J., Liu, S., Lin, S., Berube, P., Lee, L., Chen, J., Brumbaugh, J., Rigollet, P., Hochedlinger, K., Jaenisch, R., Regev, A., & Lander, E. S. (2019). Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming. *Cell*, 176(4), 928–943.e22.

Schmidt-Hieber, J. (2020). Nonparametric regression using deep neural networks with ReLU activation function. *The Annals of Statistics*, 48(4), 1875–1897.

Seguy, V., Damodaran, B. B., Flamary, R., Courty, N., Rolet, A., & Blondel, M. (2018). Large-scale optimal transport and mapping estimation. In *International Conference on Learning Representations*.

Seijo, E. & Sen, B. (2011). Nonparametric least squares estimation of a multivariate convex regression function. *The Annals of Statistics*, 39(3), 1633–1657.

Sivaprasad, S., Singh, A., Manwani, N., & Gandhi, V. (2021). The curious case of convex neural networks. In N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, & J. A. Lozano (Eds.), *Machine Learning and Knowledge Discovery in Databases. Research Track*, volume 12975 of *Lecture Notes in Computer Science*, pages 738–754.: Springer.

Song, Y., Durkan, C., Murray, I., & Ermon, S. (2021). Maximum likelihood training of score-based diffusion models. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems*, volume 34, pages 1415–1428.: Curran Associates, Inc.

Stromme, A. J. (2024). Minimum intrinsic dimension scaling for entropic optimal transport. In J. Ansari, S. Fuchs, W. Trutschnig, M. A. Lubiano, M. Á. Gil, P. Grzegorzewski, & O. Hryniewicz (Eds.), *Combining, Modelling and Analyzing Imprecision, Randomness and Dependence*, volume 1458 of *Advances in Intelligent Systems and Computing*, pages 491–499.: Springer.

Taghvaei, A. & Jalali, A. (2019). 2-Wasserstein approximation via restricted convex potentials with application to improved training for GANs. *Preprint arXiv:1902.07197*.

Tameling, C., Stoldt, S., Stephan, T., Naas, J., Jakobs, S., & Munk, A. (2021). Colocalization for super-resolution microscopy via optimal transport. *Nature Computational Science*, 1(3), 199–211.

Tan, H. Y., Mukherjee, S., Tang, J., & Schönlieb, C.-B. (2023). Data-driven mirror descent with input-convex neural networks. *SIAM Journal on Mathematics of Data Science*, 5(2), 558–587.

Telgarsky, M. (2016). Benefits of depth in neural networks. In V. Feldman, A. Rakhlin, & O. Shamir (Eds.), *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1517–1539. Columbia University, New York, New York, USA: PMLR.

Thakolkaran, P., Guo, Y., Saini, S., Peirlinck, M., Alheit, B., & Kumar, S. (2025). Can KAN CANs? Input-convex Kolmogorov-Arnold networks (KANs) as hyperelastic constitutive artificial neural networks (CANs). *Computer Methods in Applied Mechanics and Engineering*, 443, 118089.

Uscidda, T. & Cuturi, M. (2023). The Monge gap: A regularizer to learn all transport maps. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, & J. Scarlett (Eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 34709–34733.: PMLR.

Villani, C. (2009). *Optimal transport: Old and new*, volume 338 of *Grundlehren der mathematischen Wissenschaften*. Springer.

Wang, W., Ozolek, J. A., Slepčev, D., Lee, A. B., Chen, C., & Rohde, G. K. (2011). An optimal transportation approach for nuclear structure-based pathology. *IEEE Transactions on Medical Imaging*, 30(3), 621–631.

Warin, X. (2024). The GroupMax neural network approximation of convex functions. *IEEE Transactions on Neural Networks and Learning Systems*, 35(8), 11608–11612.

Warin, X. (2026). P1-KAN: An effective Kolmogorov-Arnold network with application to hydraulic valley optimization. *Neural Computing and Applications*, 38(16), 703.

Weed, J. & Bach, F. (2019). Sharp asymptotic and finite-sample rates of convergence of empirical measures in Wasserstein distance. *Bernoulli*, 25(4A), 2620–2648.

Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94, 103–114.

Supplementary Material

The supplementary material is organized as follows. Section A provides additional theoretical results on the approximation capabilities of HyCNNs. Section B collects all omitted proofs of the results stated in the main text and in Section A. Section C details the derivation of the initialization scheme for HyCNNs introduced in Section 2. Section D contains additional experimental results and implementation details (complementing Section 5), including visualizations of randomly initialized HyCNNs, further regression experiments, and additional results on neural OT map estimation. Section E presents further details to neural OT based single-cell perturbation response prediction for the preprocessed 4i single-cell dataset (Bunne et al., 2023).

A Additional Theoretical Results

In this section, we extend the approximation results to multivariate quadratic functions and a class of univariate analytic functions. The proofs are deferred to Appendix B.6.

The following theorems state the multivariate extensions of the approximation lower and upper bounds for approximating the quadratic function $\mathbf{x} \mapsto \|\mathbf{x}\|_2^2$ on $[0,1]^d$ by ReLU ICNNs and HyCNNs, respectively.

Theorem A.1. *Any (standard or leaky) ReLU ICNN $f_\theta : \mathbb{R}^d \to \mathbb{R}$ with L hidden layers and d_ℓ neurons in layer ℓ satisfies the lower bound*

$$\int_{[0,1]^d} \left| f_\theta(\mathbf{x}) - \|\mathbf{x}\|_2^2 \right| d\mathbf{x} \geq \frac{1}{16(d_1 + 2\sum_{\ell=2}^{L} d_\ell)^2} \quad \text{and}$$

$$\sup_{\mathbf{x}\in[0,1]^d} \left| f_\theta(\mathbf{x}) - \|\mathbf{x}\|_2^2 \right| \geq \frac{d}{8(d_1 + 2\sum_{\ell=2}^{L} d_\ell)^2}.$$

Theorem A.2. *For any $d, L, m \in \mathbb{N}$ with $m \geq 3$ and $\tau \geq 0$, there exists a HyCNN $f : \mathbb{R}^d \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$, L hidden layers and m neurons per layer, such that*

$$\sup_{\mathbf{x}\in[0,1]^d} \left| f(\mathbf{x}) - \|\mathbf{x}\|_2^2 \right| \leq \frac{d}{7.99} \inf_{p,q\in\mathbb{N}: pq \geq d} \left\lfloor \frac{m-1}{q} - 1 \right\rfloor^{-2\lfloor L/p \rfloor}.$$

The key conceptual insights are that also in high-dimensional settings, HyCNNs can approximate the quadratic function $\mathbf{x} \mapsto \|\mathbf{x}\|_2^2$ with an error that decays exponentially in the depth and polynomially in the width, whereas (standard or leaky) ReLU ICNNs can only achieve a polynomial decay in the width and no decay in the depth. Indeed, combining Theorem 11 of Liang & Srikant (2017) with the last inclusion in (5) shows that the exponential decay with respect to the depth is optimal, for both ReLU networks and HyCNNs.

We next extend the approximation result beyond quadratic functions to analytic functions whose Taylor coefficients of order two and higher are non-negative. The following theorem provides an approximation error bound that decays exponentially in L.

Theorem A.3. *Assume that $r > 1$ and that $f_0 : \mathbb{R} \to \mathbb{R}$ is real analytic on an open interval containing $[0, r]$, with $f_0^{(k)}(0) \geq 0$ for all $k \geq 2$. For any $L \in \mathbb{N}$, odd number $m \geq 3$ and $\tau \geq 0$, there exists a HyCNN $h : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a, b) = \overline{\sigma}_\tau(a, b)$, $\lceil \log_2(2L) \rceil L$ layers and $2Lm$ neurons per layer such that*

$$\sup_{x\in[0,1]} \left| h(x) - f_0(x) \right| \lesssim r^{-2L} + (m-1)^{-2L},$$

where the implicit constant depends only on f_0.

The target function class in Theorem A.3 is much larger than the class of monomials. In fact, it contains functions which are analytic on an open interval containing $[0, r]$ for some $r > 1$ and absolutely monotonic on $[0, 1]$, meaning that their derivatives of all orders are non-negative on $[0, 1]$. Examples include $\exp(x)$, $\sinh(x)$, $\cosh(x)$, as well as the functions $\sin^{-1}(\gamma x)$, $-\log(1-\gamma x)$ and $(1-\gamma x)^{-\alpha}$ with $\gamma \in (0, 1)$ and $\alpha > 0$.

The following proposition shows that uniform approximation of a smooth convex target function g by a differentiable convex function f yields a bound on the gradient approximation error. We write $K := [0,1]^d \subset K' := (-1, 2)^d$.

Proposition A.4. *Let $f, g\colon K' \to \mathbb{R}$ be differentiable and convex, and let g be M-smooth on K' with $M \geq 0$, i.e.,*

$$g(\mathbf{y}) \;\leq\; g(\mathbf{x}) + \langle \nabla g(\mathbf{x}),\, \mathbf{y} - \mathbf{x}\rangle + \tfrac{M}{2}\|\mathbf{y} - \mathbf{x}\|^2 \qquad \text{for all } \mathbf{x}, \mathbf{y} \in K'.$$

Then, for any $p \in [1, \infty)$,

$$\|\nabla f - \nabla g\|_{L^p(K)} \leq \|\nabla f - \nabla g\|_{L^\infty(K)} \leq 2\sqrt{M\varepsilon} + 4\varepsilon, \qquad \varepsilon := \|f - g\|_{L^\infty(K')}.$$

Hence, exponential approximation rates for smooth convex target functions yield exponential approximation rates for their gradients.

B Omitted Proofs

B.1 Technical lemmas

Lemma B.1. *Let $\overline{\sigma}\colon \mathbb{R}^2 \to \mathbb{R}$ be convex and componentwise non-decreasing. If $f : \mathbb{R}^d \to \mathbb{R}$ and $g : \mathbb{R}^d \to \mathbb{R}$ are convex functions, then $h : \mathbf{x} \mapsto \overline{\sigma}(f(\mathbf{x}), g(\mathbf{x}))$ is convex.*

Proof. For any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ and $t \in [0, 1]$, we have

$$f\big(t\mathbf{x}_1 + (1-t)\mathbf{x}_2\big) \leq tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2), \quad g\big(t\mathbf{x}_1 + (1-t)\mathbf{x}_2\big) \leq tg(\mathbf{x}_1) + (1-t)g(\mathbf{x}_2).$$

Hence, by the assumptions imposed on $\overline{\sigma}$ we get

$$\begin{aligned}
h\big(t\mathbf{x}_1 + (1-t)\mathbf{x}_2\big) &= \overline{\sigma}\Big(f\big(t\mathbf{x}_1 + (1-t)\mathbf{x}_2\big),\, g\big(t\mathbf{x}_1 + (1-t)\mathbf{x}_2\big)\Big) \\
&\leq \overline{\sigma}\big(tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2),\, tg(\mathbf{x}_1) + (1-t)g(\mathbf{x}_2)\big) \\
&= \overline{\sigma}\Big(t\big(f(\mathbf{x}_1), g(\mathbf{x}_1)\big)^\mathsf{T} + (1-t)\big(f(\mathbf{x}_2), g(\mathbf{x}_2)\big)^\mathsf{T}\Big) \\
&\leq t\overline{\sigma}\big(f(\mathbf{x}_1), g(\mathbf{x}_1)\big) + (1-t)\overline{\sigma}\big(f(\mathbf{x}_2), g(\mathbf{x}_2)\big) \\
&= th(\mathbf{x}_1) + (1-t)h(\mathbf{x}_2).
\end{aligned}$$

□

The following lemma shows that any maxout HyCNN can be approximated to arbitrary accuracy by a HyCNN with a smooth activation $\overline{\sigma}_\tau$ and the same architecture.

Lemma B.2. *For any HyCNN $h\colon \mathbb{R}^d \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$ and any $\tau, \epsilon > 0$, there exists a HyCNN $g\colon \mathbb{R}^d \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$ and the same architecture as h such that*

$$\sup_{\mathbf{x} \in \mathbb{R}^d} |g(\mathbf{x}) - h(\mathbf{x})| \leq \epsilon.$$

Proof. For $m := \max(a_1, a_2)$,

$$\overline{\sigma}_\tau(a_1, a_2) = \tau \log\left(e^{a_1/\tau} + e^{a_2/\tau}\right) = m + \tau \log\left(e^{(a_1 - m)/\tau} + e^{(a_2 - m)/\tau}\right).$$

For any fixed $\tau > 0$ and any $a_1, a_2 \in \mathbb{R}$, we have

$$\max(a_1, a_2) \leq \overline{\sigma}_\tau(a_1, a_2) \leq \max(a_1, a_2) + \tau \log 2 \leq \max(a_1, a_2) + \tau.$$

Hence, for any $s > 0$, it holds that

$$\max(a_1, a_2) \leq \overline{\sigma}_{s\tau}(a_1, a_2) = s\overline{\sigma}_\tau(a_1/s, a_2/s) \leq \max(a_1, a_2) + s\tau. \tag{7}$$

The idea of the proof is to show that, for any fixed $\tau > 0$, a HyCNN with activation function $\overline{\sigma}_\tau$ can approximate the maxout HyCNN h to arbitrary precision by appropriately rescaling the network parameters.

Formally, let L denote the number of hidden layers of h, and let d_ℓ denote the number of neurons in the ℓ-th hidden layer of h. The output of h and the hidden layers are defined by

$$h(\mathbf{x}) := V_L \mathbf{z}_L + W_L \mathbf{x} + \mathbf{b}_L, \qquad \mathbf{z}_{\ell+1} := \max\left(V_\ell^1 \mathbf{z}_\ell + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1, V_\ell^2 \mathbf{z}_\ell + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2\right),$$

for $\ell = 0, \dots, L-1$ with $\mathbf{z}_0 := \mathbf{0}_d$. We let

$$B := \max\Big(1, \|V_L\|_{\max}, \|W_L\|_{\max}, \|\mathbf{b}_L\|_{\max}, \max_{\substack{0\le\ell\le L-1\\ k\in\{1,2\}}} \big(\|V_\ell^k\|_{\max}, \|W_\ell^k\|_{\max}, \|\mathbf{b}_\ell^k\|_{\max}\big)\Big)$$

denote the maximum absolute value among these parameters. Here, $\|\cdot\|_{\max}$ denotes the maximum absolute entry of a matrix or vector.

We construct a HyCNN $g\colon \mathbb{R}^d \to \mathbb{R}$ defined by

$$g(\mathbf{x}) := M_L \mathbf{y}_L + N_L \mathbf{x} + \mathbf{c}_L, \qquad \mathbf{y}_{\ell+1} := \overline{\sigma}_\tau\left(M_\ell^1 \mathbf{y}_\ell + N_\ell^1 \mathbf{x} + \mathbf{c}_\ell^1, M_\ell^2 \mathbf{y}_\ell + N_\ell^2 \mathbf{x} + \mathbf{c}_\ell^2\right),$$

for $\ell = 0, \dots, L-1$ with $\mathbf{y}_0 := \mathbf{0}_d$, where the parameters will be chosen as follows. Fix arbitrary $s > 0$. Define

$$M_0^k := V_0^k/s, \quad N_0^k := W_0^k/s, \quad \mathbf{c}_0^k := \mathbf{b}_0^k/s, \qquad k = 1, 2.$$

Then, from

$$\mathbf{y}_1 = \overline{\sigma}_\tau\left(M_0^1 \mathbf{y}_0 + N_0^1 \mathbf{x} + \mathbf{c}_0^1, M_0^2 \mathbf{y}_0 + N_0^2 \mathbf{x} + \mathbf{c}_0^2\right) = \overline{\sigma}_\tau\left(\frac{W_0^1 \mathbf{x} + \mathbf{b}_0^1}{s}, \frac{W_0^2 \mathbf{x} + \mathbf{b}_0^2}{s}\right),$$

and (7), we get $\|s\mathbf{y}_1 - \mathbf{z}_1\|_\infty \le s\tau$. Assume that, for some $\ell \in [L-1]$, there exist parameters $(M_i^k, N_i^k, \mathbf{c}_i^k)_{i\in\{0,\dots,\ell-1\},\, k\in\{1,2\}}$ of g such that

$$\|s\mathbf{y}_\ell - \mathbf{z}_\ell\|_\infty \le s\tau \prod_{i=1}^{\ell-1} (2Bd_i)$$

with the convention that $\prod_\varnothing = 1$. Define

$$M_\ell^k := V_\ell^k, \quad N_\ell^k := W_\ell^k/s, \quad \mathbf{c}_\ell^k := \mathbf{b}_\ell^k/s, \qquad k = 1, 2.$$

Then, from

$$\begin{aligned}
\mathbf{y}_{\ell+1} &= \overline{\sigma}_\tau\left(M_\ell^1 \mathbf{y}_\ell + N_\ell^1 \mathbf{x} + \mathbf{c}_\ell^1, M_\ell^2 \mathbf{y}_\ell + N_\ell^2 \mathbf{x} + \mathbf{c}_\ell^2\right) \\
&= \overline{\sigma}_\tau\left(\frac{V_\ell^1(s\mathbf{y}_\ell) + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1}{s}, \frac{V_\ell^2(s\mathbf{y}_\ell) + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2}{s}\right),
\end{aligned}$$

(7) and the inequality $|\max(a,b) - \max(a',b')| \le \max(|a-a'|, |b-b'|)$, we get

$$\begin{aligned}
\|s\mathbf{y}_{\ell+1} - \mathbf{z}_{\ell+1}\|_\infty \le{}& \left\| s\overline{\sigma}_\tau\left(\frac{V_\ell^1(s\mathbf{y}_\ell) + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1}{s}, \frac{V_\ell^2(s\mathbf{y}_\ell) + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2}{s}\right)\right. \\
&\left. - \max\left(V_\ell^1(s\mathbf{y}_\ell) + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1, V_\ell^2(s\mathbf{y}_\ell) + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2\right)\right\|_\infty \\
&+ \left\| \max\left(V_\ell^1(s\mathbf{y}_\ell) + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1, V_\ell^2(s\mathbf{y}_\ell) + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2\right)\right. \\
&\left. - \max\left(V_\ell^1 \mathbf{z}_\ell + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1, V_\ell^2 \mathbf{z}_\ell + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2\right)\right\|_\infty \\
\le{}& s\tau + Bd_\ell s\tau \prod_{i=1}^{\ell-1} (2Bd_i) \\
\le{}& s\tau \prod_{i=1}^{\ell} (2Bd_i).
\end{aligned}$$

Hence, by induction, we obtain

$$\|s\mathbf{y}_L - \mathbf{z}_L\|_\infty \le s\tau \prod_{i=1}^{L-1} (2Bd_i).$$

By defining $M_L := sV_L$, $N_L := W_L$ and $\mathbf{c}_L := \mathbf{b}_L$, we get

$$|g(\mathbf{x}) - h(\mathbf{x})| \le Bd_L \|s\mathbf{y}_L - \mathbf{z}_L\|_\infty \le s\tau 2^{L-1} B^L \prod_{i=1}^{L} d_i.$$

Recall that $s > 0$ can be chosen arbitrarily. By choosing $0 < s \le \epsilon/(\tau 2^{L-1} B^L \prod_{i=1}^L d_i)$, we obtain the assertion. □

The entrywise non-negativity constraint imposed on the hidden-to-hidden weight matrices $V_1, V_2, \dots, V_{L-1}$ in ICNNs is necessary to ensure the input convexity. Because ICNNs form a subclass of HyCNNs, the result also shows that the input convexity of HyCNNs requires entrywise non-negativity of the matrices $V_1^1, V_1^2, V_2^1, V_2^2, \dots, V_{L-1}^1, V_{L-1}^2$.

Lemma B.3. *Consider ReLU ICNNs with $L \geq 2$ layers and assume that for an arbitrary $\ell \in \{1, \dots, L-1\}$ and an arbitrary entry of V_ℓ, the non-negativity constraint does not hold. For any $i = 1, \dots, d$, the resulting function class will then also contain the non-convex functions $f_i : [0,1]^d \to \mathbb{R}$, $f_i(x_1, \dots, x_d) = \min(1/2, 1 - x_i)$.*

Proof. Let the selected ℓ, i be denoted by ℓ^* and i^*. For arbitrary $m \in \{1, \dots, d_{\ell^*+1}\}, n \in \{1, \dots, d_{\ell^*}\}$, assume that the (m,n)-th entry of V_{ℓ^*} is allowed to be negative. We show that the right-hand side of the identity $\min(1/2, 1 - x_i) = (1/2 - (x_i - 1/2)_+)$ with $x_i \in [0,1]$ can be rewritten as an ICNN. Choose all the matrices $W_1, \dots, W_L$ as zero matrices. For an $r \in \{1, \dots, d_1\}$, setting $\mathbf{b}_0 := -\frac{1}{2}\mathbf{e}_r$ with $\mathbf{e}_r$ the r-th standard basis vector and $W_0 := (\mathbf{1}(a = r, b = i))_{a,b}$, we have that $\mathbf{z}_1 = (x_i - 1/2)_+ \mathbf{e}_r$. For $\ell \geq 1$, $p \in \{1, \dots, d_{\ell+1}\}$, and $q \in \{1, \dots, d_\ell\}$, if $V_\ell = (\mathbf{1}(a = p, b = q))_{a,b}$ and $\mathbf{b}_\ell = \mathbf{0}_{d_{\ell+1}}$, then $\mathbf{z}_\ell = \gamma \mathbf{e}_q$ with $\gamma > 0$ implies $\mathbf{z}_{\ell+1} = \gamma \mathbf{e}_p$. Thus, one can achieve that $\mathbf{z}_{\ell^*} = (x_i - 1/2)_+ \mathbf{e}_n$. Choosing $V_{\ell^*} = (-\mathbf{1}(a = m, b = n))_{a,b}$ and $\mathbf{b}_{\ell^*} = \frac{1}{2}\mathbf{e}_m$, we then have that $\mathbf{z}_{\ell^*+1} = (1/2 - (x_i - 1/2)_+)_+ \mathbf{e}_m$. By arguing for $\ell > \ell^* + 1$ as for the layers $\ell = 1, \dots, \ell^*$, one can then show that the ICNN output is $(1/2 - (x_i - 1/2)_+) = \min(1/2, 1 - x_i)$ for all $\mathbf{x} \in [0,1]^d$. □

The extension to the log-sum-exp activation function follows from the sandwich formula (7).

The composition of two convex functions is not necessarily again a convex function. The same holds for the composition of two HyCNNs. The next result reveals that specific compositions of HyCNNs can still be represented by deeper HyCNNs.

Lemma B.4 (Composition of two HyCNNs)**.** *Let $\tau \geq 0$. Consider a HyCNN $g : \mathbb{R}^d \to \mathbb{R}^p$ with L_1 hidden layers and $(d_1, \dots, d_{L_1})$ neurons per layer, specified for $\ell \in \{0, \dots, L_1 - 1\}$ by*

$$g(\mathbf{x}) := V_{L_1}\mathbf{z}_{L_1} + W_{L_1}\mathbf{x} + \mathbf{b}_{L_1}, \qquad \mathbf{z}_{\ell+1} := \mathbf{z}_{\ell+1}(\mathbf{x}) = \overline{\sigma}_\tau\left((V_\ell^k \mathbf{z}_\ell + W_\ell^k \mathbf{x} + \mathbf{b}_\ell^k)_{k \in \{1,2\}}\right)$$

with $\mathbf{z}_0 := \mathbf{0}_d$. In addition, consider another HyCNN $h : \mathbb{R}^{p+d} \to \mathbb{R}^q$ with L_2 hidden layers and $(p_1, \dots, p_{L_2})$ neurons per layer, specified for $\ell \in \{0, \dots, L_2 - 1\}$ by

$$\begin{aligned} h(\mathbf{y}, \mathbf{x}) &:= H_{L_2}\mathbf{u}_{L_2} + M_{L_2}\mathbf{y} + N_{L_2}\mathbf{x} + \mathbf{c}_{L_2}, \\ \mathbf{u}_{\ell+1} &:= \mathbf{u}_{\ell+1}(\mathbf{y}, \mathbf{x}) = \overline{\sigma}_\tau\left((H_\ell^k \mathbf{u}_\ell + M_\ell^k \mathbf{y} + N_\ell^k \mathbf{x} + \mathbf{c}_\ell^k)_{k \in \{1,2\}}\right) \end{aligned}$$

with $\mathbf{u}_0 := \mathbf{0}_p$. If the matrices $\{M_\ell^1, M_\ell^2\}_{\ell=0}^{L_2-1}$ and M_{L_2} are all componentwise non-negative, then there exists a HyCNN $f : \mathbb{R}^d \to \mathbb{R}^q$ with $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$, $L_1 + L_2$ hidden layers and $(d_1, \dots, d_{L_1}, p_1 + p, \dots, p_{L_2} + p)$ neurons per layer such that $f(\mathbf{x}) = h(g(\mathbf{x}), \mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^d$.

Proof. We construct the HyCNN f with weight matrices $\{R_\ell^k\}$, skip connection matrices $\{S_\ell^k\}$ and bias vectors $\{\mathbf{d}_\ell^k\}$ as follows. The first L_1 hidden layers exactly match those of g, i.e., for $\ell \in \{0, \dots, L_1 - 1\}$ and $k \in \{1,2\}$, we set $R_\ell^k = V_\ell^k$, $S_\ell^k = W_\ell^k$ and $\mathbf{d}_\ell^k = \mathbf{b}_\ell^k$. Choosing the matrices and bias vectors identical ensures that the output of the L_1-th hidden layer of f is exactly $\mathbf{z}_{L_1}$.

For the $(L_1 + \ell)$-th hidden layer with $\ell \in \{1, \dots, L_2\}$, we consider $p_\ell + p$ neurons, where the first p_ℓ neurons are used to compute the ℓ-th hidden layer of h and the remaining p neurons are used to propagate the output of g to the ℓ-th hidden layer of h. Concretely, for $k \in \{1,2\}$, we set

$$R_{L_1}^k = \begin{bmatrix} M_0^k V_{L_1} \\ V_{L_1} \end{bmatrix}, \qquad S_{L_1}^k = \begin{bmatrix} M_0^k W_{L_1} + N_0^k \\ W_{L_1} \end{bmatrix}, \qquad \mathbf{d}_{L_1}^k = \begin{bmatrix} M_0^k \mathbf{b}_{L_1} + \mathbf{c}_0^k \\ \mathbf{b}_{L_1} - (\tau \log 2)\,\mathbf{1}_p \end{bmatrix},$$

which yields

$$\begin{aligned} R_{L_1}^k \mathbf{z}_{L_1} + S_{L_1}^k \mathbf{x} + \mathbf{d}_{L_1}^k &= \begin{bmatrix} M_0^k V_{L_1}\mathbf{z}_{L_1} + M_0^k W_{L_1}\mathbf{x} + N_0^k \mathbf{x} + M_0^k \mathbf{b}_{L_1} + \mathbf{c}_0^k \\ V_{L_1}\mathbf{z}_{L_1} + W_{L_1}\mathbf{x} + \mathbf{b}_{L_1} - (\tau \log 2)\,\mathbf{1}_p \end{bmatrix} \\ &= \begin{bmatrix} M_0^k g(\mathbf{x}) + N_0^k \mathbf{x} + \mathbf{c}_0^k \\ g(\mathbf{x}) - (\tau \log 2)\,\mathbf{1}_p \end{bmatrix}. \end{aligned}$$

Using $\overline{\sigma}_\tau(a - \tau\log 2, a - \tau\log 2) = a$, we have

$$\overline{\sigma}_\tau\left(\left(R^k_{L_1}\mathbf{z}_{L_1} + S^k_{L_1}\mathbf{x} + \mathbf{d}^k_{L_1}\right)_{k\in\{1,2\}}\right) = \begin{bmatrix}\overline{\sigma}_\tau\left(\left(M^k_0 g(\mathbf{x}) + N^k_0\mathbf{x} + \mathbf{c}^k_0\right)_{k\in\{1,2\}}\right)\\ g(\mathbf{x})\end{bmatrix} = \begin{bmatrix}\mathbf{u}_1\big(g(\mathbf{x}),\mathbf{x}\big)\\ g(\mathbf{x})\end{bmatrix}.$$

Then, for $\ell \in \{1,\dots,L_2-1\}$ and $k\in\{1,2\}$, we set weight matrices, skip connection matrices and bias vectors as

$$R^k_{L_1+\ell} = \begin{bmatrix}H^k_\ell & M^k_\ell\\ \mathbf{0}_{p\times p_\ell} & I_p\end{bmatrix}, \qquad S^k_{L_1+\ell} = \begin{bmatrix}N^k_\ell\\ \mathbf{0}_{p\times d}\end{bmatrix}, \qquad \mathbf{d}^k_{L_1+\ell} = \begin{bmatrix}\mathbf{c}^k_\ell\\ -(\tau\log 2)\,\mathbf{1}_p\end{bmatrix},$$

which yields

$$\begin{aligned}&\overline{\sigma}_\tau\left(\left(R^k_{L_1+\ell}\begin{bmatrix}\mathbf{u}_\ell\big(g(\mathbf{x}),\mathbf{x}\big)\\ g(\mathbf{x})\end{bmatrix} + S^k_{L_1+\ell}\mathbf{x} + \mathbf{d}^k_{L_1+\ell}\right)_{k\in\{1,2\}}\right)\\ &= \overline{\sigma}_\tau\left(\left(\begin{bmatrix}H^k_\ell\mathbf{u}_\ell\big(g(\mathbf{x}),\mathbf{x}\big) + M^k_\ell g(\mathbf{x}) + N^k_\ell\mathbf{x} + \mathbf{c}^k_\ell\\ g(\mathbf{x}) - (\tau\log 2)\mathbf{1}_p\end{bmatrix}\right)_{k\in\{1,2\}}\right)\\ &= \begin{bmatrix}\mathbf{u}_{\ell+1}\big(g(\mathbf{x}),\mathbf{x}\big)\\ g(\mathbf{x})\end{bmatrix}.\end{aligned}$$

Finally, for $\ell = L_1 + L_2$, we set

$$R_{L_1+L_2} = \begin{bmatrix}H_{L_2} & M_{L_2}\end{bmatrix}, \qquad S_{L_1+L_2} = N_{L_2}, \qquad \mathbf{d}_{L_1+L_2} = \mathbf{c}_{L_2}.$$

By construction, the last hidden layer of f outputs

$$\begin{bmatrix}\mathbf{u}_{L_2}\big(g(\mathbf{x}),\mathbf{x}\big)\\ g(\mathbf{x})\end{bmatrix}.$$

Hence,

$$f(\mathbf{x}) = H_{L_2}\mathbf{u}_{L_2}\big(g(\mathbf{x}),\mathbf{x}\big) + M_{L_2}g(\mathbf{x}) + N_{L_2}\mathbf{x} + \mathbf{c}_{L_2} = h\big(g(\mathbf{x}),\mathbf{x}\big).$$

In addition, since the matrices $\{R^1_\ell, R^2_\ell\}_{\ell=0}^{L_1+L_2-1}$ and $R_{L_1+L_2}$ are all componentwise non-negative, we obtain the assertion. □

B.2 Proofs for limited expressivity of ICNNs

Proof of Proposition 3.1. Consider an ICNN architecture with L hidden layers, d_ℓ neurons in layer ℓ, and activation function given either by the ReLU $\sigma(x) = \max(x,0)$ or the leaky ReLU $\sigma(x) = \max(x,\alpha x)$ with $\alpha\in(0,1)$. For $\ell\in\{0,\dots,L\}$ and $j\in[d_\ell]$, we denote by W^j_ℓ, V^j_ℓ, and b^j_ℓ the j-th row of the respective weight matrices W_ℓ, V_ℓ, and the bias $\mathbf{b}_\ell$.

For $\ell = 1$, each hidden neuron $z_{1,j} = \sigma(W^j_0 x + b^j_0)$ for $j\in[d_1]$ can be viewed as a piecewise-affine function in x with at most one kink at the location where $x\mapsto W^j_0 x + b^j_0$ crosses zero. Denote by $\mathcal{K}_1$ the set of kink locations of the first layer.

For $\ell = 2$, each function $x\mapsto z_{2,j} = \sigma(V^j_1\mathbf{z}_1 + W^j_1 x + b^j_1)$ for $j\in[d_2]$ is also a piecewise-affine function of the input x with kink locations being a subset of the kink locations of the previous layer and the locations of the roots of $x\mapsto V^h_1\mathbf{z}_1 + W^h_1 x + b^h_1$. Since all elements of V^h_1 are non-negative and $\mathbf{z}_1$ is convex as a function of the input x, it follows that $x\mapsto V^h_1\mathbf{z}_1 + W^h_1 x + b^h_1$ is again a convex function and therefore has at most two roots. Thus, the set of kink locations of the second layer $\mathcal{K}_2$ satisfies $|\mathcal{K}_2|\le|\mathcal{K}_1| + 2d_2 \le d_1 + 2d_2$.

Likewise, by induction, it follows that the set of kink locations of the ℓ-th hidden layer $\mathcal{K}_\ell$ satisfies $|\mathcal{K}_\ell|\le|\mathcal{K}_{\ell-1}|+2d_\ell \le d_1+2d_2+\cdots+2d_\ell$. Finally, for the output layer, note that $f(x) = V_L\mathbf{z}_L + W_L x + b_L$ is piecewise-affine with kink locations included in the kink locations of the last hidden layer. Thus, the set of kink locations of f is $\subseteq\mathcal{K}_L$ and by the induction argument, $|\mathcal{K}_L|\le d_1 + 2d_2 + \cdots + 2d_L$. □

The proof of Theorem 3.2 is based on the following bound on approximation of quadratic functions by affine functions.

Lemma B.5. *For any real numbers $a < b$,*

$$\inf_{\ell \text{ affine}} \int_a^b \left|x^2 - \ell(x)\right| dx = \frac{(b-a)^3}{16},$$
$$\inf_{\ell \text{ affine}} \sup_{x\in[a,b]} |x^2 - \ell(x)| = \frac{(b-a)^2}{8}.$$

The second claim of the previous result has been previously shown in Lemma 4.5 of Eckle & Schmidt-Hieber (2019). For the sake of completeness, we include a proof below.

Proof. Substitute $t := (x-m)/h \in [-1,1]$ with $m = (a+b)/2$ and $h = (b-a)/2$. There exists a one-to-one relationship between lines $\ell(x) = \alpha x + \beta$ and real parameters c, d such that $x^2 - \ell(x) = h^2\big(t^2 - ct - d\big)$. Hence,

$$\inf_{\ell \text{ affine}} \int_a^b \left|x^2 - \ell(x)\right| dx = h^3 \inf_{c,d} \int_{-1}^1 |t^2 - ct - d|\, dt. \tag{8}$$

For any real numbers c, d, and any $t \in [-1,1]$, it holds

$$\left|t^2 - d\right| = \left|\frac{(t^2 - ct - d) + (t^2 + ct - d)}{2}\right| \le \frac{|t^2 - ct - d| + |t^2 + ct - d|}{2}.$$

Consequently, it follows

$$\begin{aligned}\int_{-1}^1 |t^2 - ct - d|\, dt &= \int_0^1 \left(|t^2 - ct - d| + |t^2 + ct - d|\right) dt \\ &\ge 2\int_0^1 |t^2 - d|\, dt \\ &= \int_{-1}^1 |t^2 - d|\, dt.\end{aligned}$$

Thus, the linear term ct cannot improve the L^1 error, the infimum in (8) is attained for $c = 0$, and

$$\inf_{c,d} \int_{-1}^1 |t^2 - ct - d|\, dt = \inf_d \int_{-1}^1 |t^2 - d|\, dt = 2\inf_d \int_0^1 |t^2 - d|\, dt = \frac{1}{2},$$

where the equality is attained for $d = 1/4$. Combined with (8), we obtain

$$\inf_{\ell \text{ affine}} \int_a^b \left|x^2 - \ell(x)\right| dx = \left(\frac{b-a}{2}\right)^3 \inf_{c,d} \int_{-1}^1 |t^2 - ct - d|\, dt = \frac{(b-a)^3}{16}.$$

To prove the second claim of the lemma, (8) can be replaced by

$$\inf_{\ell \text{ affine}} \sup_{x\in[a,b]} |x^2 - \ell(x)| = h^2 \inf_{c,d} \sup_{t\in[-1,1]} \left|t^2 - ct - d\right|. \tag{9}$$

Using that for any $t \in [-1,1]$,

$$\begin{aligned}\left|t^2 - d\right| &= \left|\frac{(t^2 - ct - d) + (t^2 + ct - d)}{2}\right| \\ &\le \frac{|t^2 - ct - d| + |t^2 + ct - d|}{2} \\ &\le \max\{|t^2 - ct - d|, |t^2 + ct - d|\}\end{aligned}$$

yields $\sup_{t\in[-1,1]} |t^2 - d| \le \sup_{t\in[-1,1]} |t^2 - ct - d|$. Thus, the linear term ct cannot improve the uniform error, and the infimum in (9) is attained for $c = 0$. Since t^2 takes all values in $[0,1]$ on $[-1,1]$, we have $\sup_{t\in[-1,1]} |t^2 - d| = \max\{|0-d|, |1-d|\}$, which is minimized at $d = \frac{1}{2}$ with value $\frac{1}{2}$. Therefore,

$$\inf_{\ell \text{ affine}} \sup_{x\in[a,b]} |x^2 - \ell(x)| = h^2 \cdot \frac{1}{2} = \frac{(b-a)^2}{8}.$$

□

Proof of Theorem 3.2. Let $g \in \mathcal{G}_k$. Then there exists a partition $0 = t_0 < t_1 < \cdots < t_m = 1$ with $m \le k$ such that g is affine on each $[t_{j-1}, t_j]$. Let $L_j := t_j - t_{j-1}$ and $L_{\max} := \max_j L_j$. Since $\sum_j L_j = 1$, we have $L_{\max} \ge 1/m \ge 1/k$. On the interval $I = [t_{j-1}, t_j]$ with $L_j = L_{\max}$ the function g is affine, hence by Lemma B.5

$$\sup_{x\in[0,1]} |g(x) - x^2| \ge \sup_{x\in I} |g(x) - x^2| \ \ge \ \inf_{\ell \text{ affine}} \sup_{x\in I} |\ell(x) - x^2| = \frac{L_{\max}^2}{8} \ \ge \ \frac{1}{8k^2}.$$

On the other hand, Lemma B.5 and Jensen's inequality applied to $z \mapsto z^3$ yield

$$\int_0^1 \left|g(x) - x^2\right| dx \ge \sum_{k=1}^{m} \left(\inf_{\ell \text{ affine}} \int_{t_{k-1}}^{t_k} \left|x^2 - \ell(x)\right| dx \right) = \frac{1}{16} \sum_{j=1}^{m} L_j^3 \ge \frac{1}{16m^2} \ge \frac{1}{16k^2}.$$

Since these bounds hold for every $g \in \mathcal{G}_k$, the assertion follows by taking the infimum over $g \in \mathcal{G}_k$. By combining these results with Proposition 3.1, we obtain the remaining assertions. □

Remark B.6 (Lower bounds for higher order monomials). For higher order monomials $x \in [0,1] \mapsto x^n$ with $n = 3, \dots,$ similar lower bounds as for the quadratic function hold, though the proof is more involved. E.g., McClure (1975, Equation (6.17)) asserts for $n \ge 2$ that

$$\liminf_{k\to\infty} k^2 \inf_{g\in\mathcal{G}_k} \int_0^1 |g(x) - x^n| \, dx \ge C_n > 0$$

where $\mathcal{G}_k$ is the set of real-valued piecewise-affine, convex functions and C_n only depends on n.

B.3 Proof for approximation of quadratic functions with HyCNNs

In this section, we provide the proof of Theorem 3.3. We first establish the following lemma, which will be used to prove the main result. The proof of Theorem 3.3 then follows by applying Lemma B.2.

Lemma B.7 (Quadratic Function Approximation with maxout unit)**.** *For any $L \in \mathbb{N}$ and even numbers $d_1, \dots, d_L$, there exists a HyCNN $h\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L hidden layers, and $(d_1, \dots, d_L)$ neurons per layer such that*

$$\sup_{x\in[0,1]} |h(x) - x^2| \le \frac{1}{8 \prod_{\ell=1}^{L} d_\ell^2}.$$

To prove Lemma B.7, we first establish the following lemma, writing $x_+ := \max(x, 0)$.

Lemma B.8. *For any $m = 1, 2, \dots,$*

$$\sup_{x\in[0,1]} \left| \left(-\frac{1}{8m^2} + \frac{x}{m} + \frac{2}{m} \sum_{k=1}^{m-1} \left(x - \frac{k}{m}\right)_+ \right) - x^2 \right| \le \frac{1}{8m^2}.$$

Proof. We fix $j \in [m]$. For any $x \in [\frac{j-1}{m}, \frac{j}{m}]$, observe that

$$-\frac{1}{8m^2} + \frac{x}{m} + \frac{2}{m} \sum_{k=1}^{m-1} \left(x - \frac{k}{m}\right)_+ = -\frac{1}{8m^2} + \frac{x}{m} + \frac{2}{m} \sum_{i=1}^{j-1} \left(x - \frac{i}{m}\right)_+ = \frac{2j-1}{m} x - \frac{j(j-1)}{m^2} - \frac{1}{8m^2}.$$

Hence,

$$\sup_{x\in[\frac{j-1}{m}, \frac{j}{m}]} \left| \frac{2j-1}{m} x - \frac{j(j-1)}{m^2} - \frac{1}{8m^2} - x^2 \right| = \sup_{x\in[\frac{j-1}{m}, \frac{j}{m}]} \left| -\left(x - \frac{j - \frac{1}{2}}{m}\right)^2 + \frac{1}{8m^2} \right| \le \frac{1}{8m^2}.$$

Since the inequality holds for arbitrary $j \in [m]$, we obtain the assertion. □

Proof of Lemma B.7. See Figure 6 for an illustration of the proof. We define $D_\ell := \prod_{i=1}^{\ell} d_i$ for $\ell \in [L]$. The construction of the network is done by induction with respect to the layer ℓ via the following three steps:

(i) Let $\mathbf{z}_\ell(x) = (z_{\ell,j}(x))_{j=0}^{d_\ell - 1}$ be the values of the d_ℓ hidden neurons in the ℓ-th hidden layer.

(ii) We define three functions $u_\ell(x), \phi_\ell(x)$ and $\psi_\ell(x)$, which are non-negative affine combinations of $\mathbf{z}_\ell(x)$.

(iii) We define $\mathbf{z}_{\ell+1}(x) = (z_{\ell+1,j}(x))_{j=0}^{d_{\ell+1}-1}$ via (14) and (15). Since $u_\ell(x), \phi_\ell(x)$ and $\psi_\ell(x)$ are non-negative affine combinations of $\mathbf{z}_\ell(x)$, $\mathbf{z}_{\ell+1}(x)$ is of the form $\max(V_\ell^1 \mathbf{z}_\ell(x) + \mathbf{w}_\ell^1 x + \mathbf{b}_\ell^1, V_\ell^2 \mathbf{z}_\ell(x) + \mathbf{w}_\ell^2 x + \mathbf{b}_\ell^2)$ with element-wise non-negative V_ℓ^1 and V_ℓ^2.

We begin the induction with the first hidden layer $\ell = 1$. In the first hidden layer, we construct $d_1 - 1$ functions $\mathbf{z}_1(x) = (z_{1,j}(x))_{j=1}^{d_1-1}$ of the form

$$z_{1,j}(x) := \left(x - \frac{j}{d_1}\right)_+ \quad \text{for } j \in [d_1 - 1]. \tag{10}$$

Only $d_1 - 1$ neurons are used here. We consider three non-negative affine combinations of $\mathbf{z}_1(x)$:

$$\begin{aligned}
u_1(x) &:= \sum_{k=1}^{d_1-1} \frac{2}{d_1}\left(x - \frac{k}{d_1}\right)_+, \\
\phi_1(x) &:= \sum_{k=1}^{d_1-1} \big(a_1 \mathbb{1}(k \text{ odd}) + b\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{d_1}\right)_+, \\
\psi_1(x) &:= \sum_{k=1}^{d_1-1} \big(b\mathbb{1}(k \text{ odd}) + a_1\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{d_1}\right)_+
\end{aligned}$$

with $a_1 := 4/d_1$ and $b := 4/(LD_L)$. This proves step (ii) for $\ell = 1$.

Let

$$a_{\ell+1} := \frac{a_\ell - (d_{\ell+1} - 1)b}{d_{\ell+1}}.$$

This implies that $a_\ell \geq (L + 1 - \ell)(\prod_{i=\ell+1}^{L} d_i)b \geq b$ for every $\ell \in [L]$. Indeed, this is true for $\ell = 1$ as $a_1 = L(\prod_{i=2}^{L} d_i)b$. Moreover, if $a_\ell \geq (L + 1 - \ell)(\prod_{i=\ell+1}^{L} d_i)b$, then

$$a_{\ell+1} \geq \frac{a_\ell - d_{\ell+1}b}{d_{\ell+1}} \geq \frac{(L-\ell)(\prod_{i=\ell+1}^{L} d_i)b}{d_{\ell+1}} = (L-\ell)\Big(\prod_{i=\ell+2}^{L} d_i\Big)b.$$

We now describe the induction step $\ell \to \ell + 1$, assuming that for a given $\ell \in [L-1]$,

$$u_\ell(x) := \sum_{k=1}^{D_\ell-1} \frac{2}{D_\ell}\left(x - \frac{k}{D_\ell}\right)_+, \tag{11}$$

$$\phi_\ell(x) := \sum_{k=1}^{D_\ell-1} \big(a_\ell \mathbb{1}(k \text{ odd}) + b\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{D_\ell}\right)_+, \tag{12}$$

$$\psi_\ell(x) := \sum_{k=1}^{D_\ell-1} \big(b\mathbb{1}(k \text{ odd}) + a_\ell\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{D_\ell}\right)_+, \tag{13}$$

are non-negative affine combinations of the values of the neurons in the ℓ-th hidden layer $\mathbf{z}_\ell(x) = (z_{\ell,j}(x))_{j=0}^{d_\ell-1}$.

In the $(\ell + 1)$-st hidden layer, we can construct $d_{\ell+1}$ neurons with

$$z_{\ell+1,0}(x) := \max\big(u_\ell(x), 0\big) = u_\ell(x) = \sum_{k=1}^{D_\ell-1} \frac{2}{D_\ell}\left(x - \frac{k}{D_\ell}\right)_+ \tag{14}$$

and

$$z_{\ell+1,j}(x) := \max\left(\phi_\ell(x), \psi_\ell(x) + \frac{a_\ell - b}{2}x - \frac{(a_\ell - b)j}{2D_{\ell+1}}\right). \tag{15}$$

for $j \in [d_{\ell+1} - 1]$. The following two lemmas complete the induction step. Their proofs are given below.

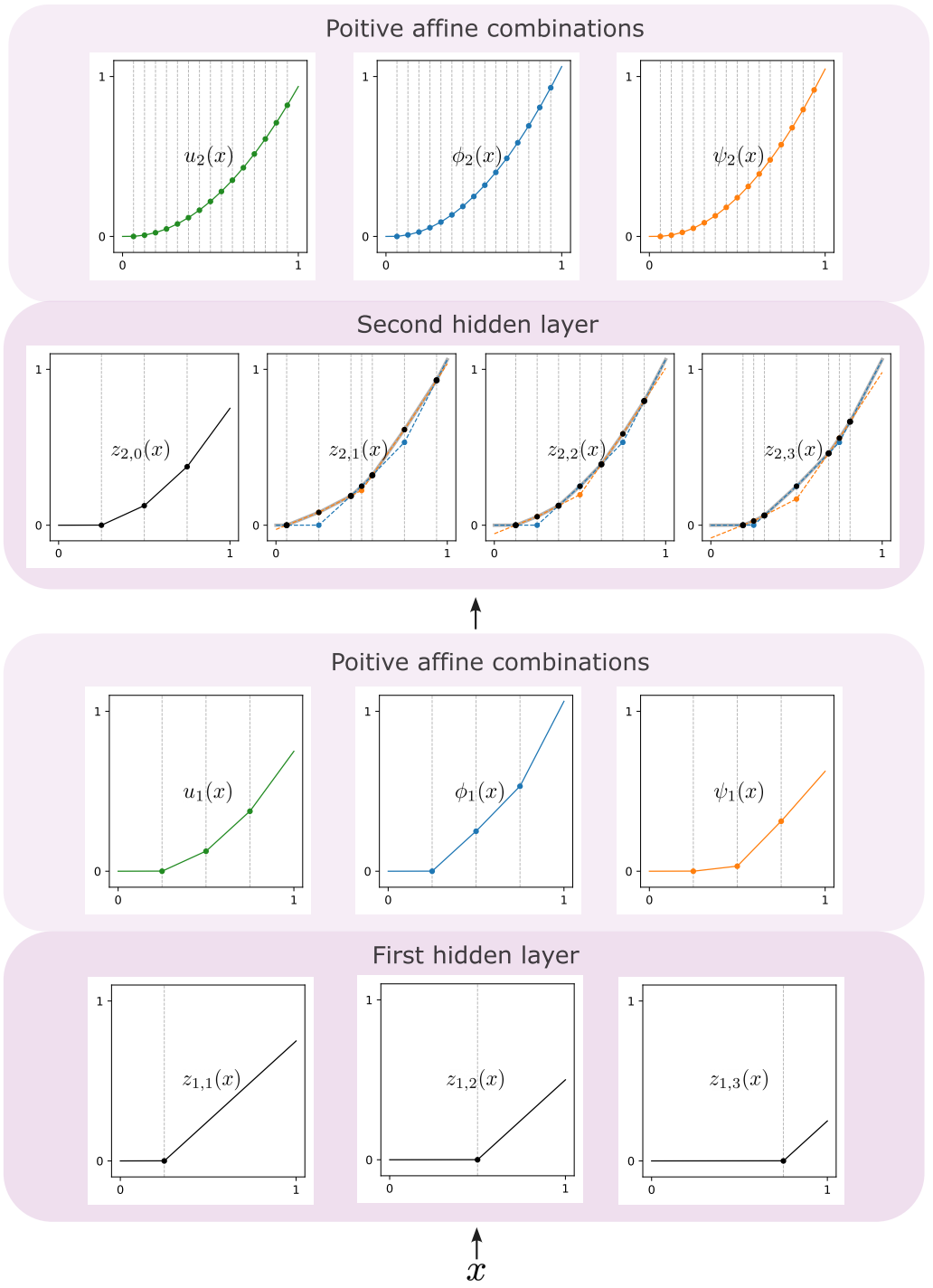

Figure 6: **Illustration of the proof of Lemma B.7** for $L = 2$ and $(d_1, d_2) = (4, 4)$. $u_\ell(x)$, $\phi_\ell(x)$ and $\psi_\ell(x)$ are defined by (11), (12) and (13), respectively. From (14) and (15), $z_{2,0}(x) = u_1(x)$ and $z_{2,j}(x) = \max(\phi_1(x), \psi_1(x) + \frac{7}{16}x - \frac{7j}{256})$ for $j \in \{1, 2, 3\}$. Gray vertical lines are drawn at the kink locations of the functions.

Lemma B.9. *For each $j \in \{1, \dots, d_{\ell+1} - 1\}$, $z_{\ell+1,j}(x)$ defined by (15) equals*

$$\sum_{k=1}^{D_\ell - 1} b\Big(x - \frac{k}{D_\ell}\Big)_+ + \sum_{k=0}^{D_\ell - 1} \frac{a_\ell - b}{2}\left(\mathbb{1}(k \text{ even})\Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ + \mathbb{1}(k \text{ odd})\Big(x - \frac{k}{D_\ell} - \frac{d_{\ell+1} - j}{D_{\ell+1}}\Big)_+\right).$$

Lemma B.10. *If $d_{\ell+1}$ is even, then, the functions $u_{\ell+1}$, $\phi_{\ell+1}$, and $\psi_{\ell+1}$, defined by*

$$\begin{aligned}
u_{\ell+1}(x) &:= \sum_{k=1}^{D_{\ell+1}-1} \frac{2}{D_{\ell+1}}\left(x - \frac{k}{D_{\ell+1}}\right)_+, \\
\phi_{\ell+1}(x) &:= \sum_{k=1}^{D_{\ell+1}-1} \big(a_{\ell+1}\mathbb{1}(k \text{ odd}) + b\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{D_{\ell+1}}\right)_+, \\
\psi_{\ell+1}(x) &:= \sum_{k=1}^{D_{\ell+1}-1} \big(b\mathbb{1}(k \text{ odd}) + a_{\ell+1}\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{D_{\ell+1}}\right)_+,
\end{aligned}$$

are non-negative affine combinations of $(z_{\ell+1,j}(x))_{j=0}^{d_{\ell+1}-1}$, defined by (14) and (15).

Hence, by Lemma B.10 and the induction argument, the function

$$u_L(x) := \sum_{k=1}^{D_L - 1} \frac{2}{D_L}\left(x - \frac{k}{D_L}\right)_+$$

is a non-negative affine combination of the last hidden layer $\mathbf{z}_L(x)$. The final output of the network is then given by

$$f(x) := -\frac{1}{8D_L^2} + \frac{x}{D_L} + u_L(x) = -\frac{1}{8D_L^2} + \frac{x}{D_L} + \frac{2}{D_L}\sum_{k=1}^{D_L - 1}\left(x - \frac{k}{D_L}\right)_+. \tag{16}$$

Finally, by applying Lemma B.8, we have $|x^2 - f(x)| \le 1/(8D_L^2)$ and obtain the assertion. □

Remark B.11. From (10), (14) and (15), we observe that not only elements of $\{V_\ell^1\}_{\ell=0}^{L-1}$ and V_L, but also $\{\mathbf{w}_\ell^1\}_{\ell=0}^{L-1}$ and w_L in the network are non-negative. This property is exploited in the approximation of monomial functions in Appendix B.4.

Proof of Lemma B.9. We fix $\ell \in \mathbb{N}$, $j \in [d_{\ell+1} - 1]$ and recall that $D_\ell = \prod_{i=1}^{\ell} d_i$ and

$$\begin{aligned}
\phi_\ell(x) &= \sum_{k=1}^{D_\ell - 1} \big(a_\ell \mathbb{1}(k \text{ odd}) + b\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{D_\ell}\right)_+, \\
\psi_\ell(x) &= \sum_{k=1}^{D_\ell - 1} \big(b\mathbb{1}(k \text{ odd}) + a_\ell\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{D_\ell}\right)_+.
\end{aligned}$$

Further, we write $\Delta := \Delta_\ell = a_\ell - b \ge 0$ and

$$\mathcal{L}(x) := -\frac{\Delta j}{2D_\ell d_{\ell+1}} + \frac{\Delta}{2}x, \qquad F(x) := \max\Big(\phi_\ell(x),\, \mathcal{L}(x) + \psi_\ell(x)\Big).$$

We prove that on $[0, 1]$

$$\begin{aligned}
F(x) = &\sum_{k=1}^{D_\ell - 1} b\Big(x - \frac{k}{D_\ell}\Big)_+ \\
&+ \sum_{k=0}^{D_\ell - 1} \frac{\Delta}{2}\left(\mathbb{1}(k \text{ even})\Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ + \mathbb{1}(k \text{ odd})\Big(x - \frac{k}{D_\ell} - \frac{d_{\ell+1} - j}{D_{\ell+1}}\Big)_+\right).
\end{aligned}$$

The proof is divided into multiple steps.

Step 1: Slopes and switching locations. Define

$$\begin{aligned}
t(x) &:= \mathcal{L}(x) + \psi_\ell(x) - \phi_\ell(x) \\
&= -\frac{\Delta j}{2D_\ell d_{\ell+1}} + \frac{\Delta}{2}x + \sum_{k=1}^{D_\ell - 1}\big(-\Delta\mathbb{1}(k \text{ odd}) + \Delta\mathbb{1}(k \text{ even})\big)\left(x - \frac{k}{D_\ell}\right)_+.
\end{aligned}$$

Then, t is a continuous and piecewise affine function. Moreover, on each open interval $I_k := (\frac{k}{D_\ell}, \frac{k+1}{D_\ell})$ for $k = 0, \dots, D_\ell - 1$,

$$t'(x) = \frac{\Delta}{2} + \sum_{i=1}^{k} (-1)^i \Delta = (-1)^k \frac{\Delta}{2}. \tag{17}$$

We have $t(0) = -\Delta j (2 D_\ell d_{\ell+1})^{-1} < 0$. Using (17) and induction over $k = 0, 1, \dots, D_\ell - 1$, it follows that

$$t\Big(\frac{k}{D_\ell}\Big) = \begin{cases} -\frac{j}{d_{\ell+1}} \frac{\Delta}{2D_\ell}, & k \text{ even}, \\ (1 - \frac{j}{d_{\ell+1}}) \frac{\Delta}{2D_\ell}, & k \text{ odd}. \end{cases}$$

Thus on every I_k the continuous piecewise affine function t changes sign exactly once at

$$x_k := \begin{cases} \frac{k}{D_\ell} + \frac{j}{d_{\ell+1}} \frac{1}{D_\ell}, & k \text{ even}, \\ \frac{k}{D_\ell} + \frac{d_{\ell+1} - j}{d_{\ell+1}} \frac{1}{D_\ell}, & k \text{ odd}. \end{cases}$$

Consequently, on I_k we have

$$F = \begin{cases} \phi_\ell & \text{on } (0, x_0) \cup (x_1, x_2) \cup (x_3, x_4) \cup \ldots \cup (x_{D_\ell - 1}, 1) \\ \mathcal{L} + \psi_\ell & \text{on } (x_0, x_1) \cup (x_2, x_3) \cup \ldots \cup (x_{D_\ell - 2}, x_{D_\ell - 1}). \end{cases} \tag{18}$$

In particular, F is piecewise-affine on $[0, 1]$ with kinks at

$$\Big\{ \tfrac{k}{D_\ell} : k = 1, \dots, D_\ell - 1 \Big\} \cup \Big\{ x_k : k = 0, \dots, D_\ell - 1 \Big\}. \tag{19}$$

In addition, $F(0) = 0$ and $F'(0) = 0$.

Step 2: Jump heights of the derivative. For a real-valued function $f : \mathbb{R} \to \mathbb{R}$ which has right and left derivatives, we define the jump height of f' as

$$[\![f]\!](x) := \lim_{h \downarrow 0} \frac{f(x+h) - f(x)}{h} - \lim_{h \uparrow 0} \frac{f(x+h) - f(x)}{h}.$$

At each coarse grid point $\frac{k}{D_\ell}$ for $k \in [D_\ell - 1]$, we pass from I_{k-1} to I_k. By (18) the jump height of F' at $\frac{k}{D_\ell} \in (x_{k-1}, x_k)$ is

$$[\![F]\!]\Big(\frac{k}{D_\ell}\Big) = \begin{cases} [\![\phi_\ell]\!](\frac{k}{D_\ell}) = b, & k \text{ even}, \\ [\![\mathcal{L} + \psi_\ell]\!](\frac{k}{D_\ell}) = b, & k \text{ odd}. \end{cases} \tag{20}$$

At each kink x_k the active branch switches, and by (17) the jump height of F' equals

$$[\![F]\!](x_k) = (-1)^k \big((\mathcal{L} + \psi_\ell)' - (\phi_\ell)' \big)\big|_{x_k} = \frac{\Delta}{2}. \tag{21}$$

Since every jump height of F' on $[0, 1]$ is non-negative, F is convex.

Step 3: Hinge representation. Let G be convex, piecewise-affine on $[0, 1]$ with $G(0) = G'(0) = 0$, and let $\{\delta_m\}$ be its kink locations with jump height $\alpha_m > 0$ of G' at δ_m. Then $G'(x) = \sum_m \alpha_m \mathbb{1}(x > \delta_m)$ on $x \in [0, 1] \setminus \{\delta_m\}$, hence

$$G(x) = \int_0^x G'(u)\, du = \sum_m \alpha_m \, (x - \delta_m)_+ .$$

Applying this to $G = F$ with kink locations obtained in (19) and jump heights obtained in (20) and (21) finally yields

$$F(x) = \sum_{k=1}^{D_\ell - 1} b\Big(x - \frac{k}{D_\ell}\Big)_+ + \sum_{k=0}^{D_\ell - 1} \frac{\Delta}{2} \left(\mathbb{1}(k \text{ even}) \Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big) + \mathbb{1}(k \text{ odd}) \Big(x - \frac{k}{D_\ell} - \frac{d_{\ell+1} - j}{D_{\ell+1}}\Big) \right).$$

□

Proof of Lemma B.10. Lemma B.9 states that

$$z_{\ell+1,j}(x) = \sum_{k=1}^{D_\ell-1} b\Big(x - \frac{k}{D_\ell}\Big)_+ + \sum_{k=0}^{D_\ell-1} \frac{\Delta_\ell}{2}\left(\mathbb{1}(k \text{ even})\Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ + \mathbb{1}(k \text{ odd})\Big(x - \frac{k}{D_\ell} - \frac{d_{\ell+1}-j}{D_{\ell+1}}\Big)_+\right).$$

We define constants $r_1, \dots, r_{d_{\ell+1}-1}$ as

$$r_j := \frac{2a_{\ell+1}}{\Delta_\ell}\mathbb{1}(j \text{ odd}) + \frac{2b}{\Delta_\ell}\mathbb{1}(j \text{ even}). \tag{22}$$

By assumption $d_{\ell+1}$ is even, $a_{\ell+1} = (a_\ell - (d_{\ell+1}-1)b)/d_{\ell+1}$, and therefore

$$\sum_{j=1}^{d_{\ell+1}-1} r_j = \frac{d_{\ell+1}}{2}\,\frac{2a_{\ell+1}}{\Delta_\ell} + \frac{d_{\ell+1}-2}{2}\,\frac{2b}{\Delta_\ell} = \frac{a_\ell - (d_{\ell+1}-1)b + (d_{\ell+1}-2)b}{\Delta_\ell} = 1. \tag{23}$$

It follows

$$\begin{aligned}
\sum_{j=1}^{d_{\ell+1}-1} r_j\, z_{\ell+1}^{(j)}(x) &= \sum_{k=1}^{D_\ell-1} b\Big(x - \frac{k}{D_\ell}\Big)_+ \\
&\quad + \sum_{k=0}^{D_\ell-1}\sum_{j=1}^{d_{\ell+1}-1} \frac{r_j\Delta_\ell}{2}\left(\mathbb{1}(k \text{ even})\Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ + \mathbb{1}(k \text{ odd})\Big(x - \frac{k}{D_\ell} - \frac{d_{\ell+1}-j}{D_{\ell+1}}\Big)_+\right) \\
&= \sum_{k=1}^{D_\ell-1} b\Big(x - \frac{k}{D_\ell}\Big)_+ + \sum_{k=0}^{D_\ell-1}\sum_{j=1}^{d_{\ell+1}-1} \frac{r_j\Delta_\ell}{2}\Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ \\
&= \sum_{k=1}^{D_\ell-1} b\Big(x - \frac{k}{D_\ell}\Big)_+ + \sum_{k=0}^{D_\ell-1}\sum_{j=1}^{d_{\ell+1}-1} \Big(a_{\ell+1}\mathbb{1}(j \text{ odd}) + b\mathbb{1}(j \text{ even})\Big)\Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ \\
&= \sum_{k=1}^{D_\ell-1} b\Big(x - \frac{kd_{\ell+1}}{D_{\ell+1}}\Big)_+ + \sum_{k=0}^{D_\ell-1}\sum_{j=1}^{d_{\ell+1}-1} \Big(a_{\ell+1}\mathbb{1}(j \text{ odd}) + b\mathbb{1}(j \text{ even})\Big)\Big(x - \frac{kd_{\ell+1}+j}{D_{\ell+1}}\Big)_+ \\
&= \sum_{q=1}^{D_{\ell+1}-1} \Big(a_{\ell+1}\mathbb{1}(q \text{ odd}) + b\mathbb{1}(q \text{ even})\Big)\Big(x - \frac{q}{D_{\ell+1}}\Big)_+ \\
&=: u_{\ell+1}^{(1)}(x).
\end{aligned}$$

We used (23) and (22) for the first and third equalities, the second equality uses that $d_{\ell+1}$ is even, and the fourth equality expands $q = kd_{\ell+1} + j$. Since $d_{\ell+1}$ is even, q is even if and only if j is even. Hence $u_{\ell+1}^{(1)}(x)$ is a non-negative affine combination of $(z_{\ell+1}^{(j)}(x))_{j=0}^{d_{\ell+1}-1}$.

Similarly, define

$$s_0 := \frac{D_\ell}{2}\Big(a_{\ell+1} - b\sum_{j=1}^{d_{\ell+1}-1} s_j\Big), \quad \text{and} \quad s_j := \frac{2b}{\Delta_\ell}\mathbb{1}(j \text{ odd}) + \frac{2a_{\ell+1}}{\Delta_\ell}\mathbb{1}(j \text{ even}), \quad j = 1, \dots, d_{\ell+1}-1.$$

From $\sum_{j=1}^{d_{\ell+1}-1} s_j \le \sum_{j=1}^{d_{\ell+1}-1} r_j = 1$, we get $s_0 \ge 0$. Moreover,

$$\frac{2s_0}{D_\ell} + b\sum_{j=1}^{d_{\ell+1}-1} s_j = a_{\ell+1}, \tag{24}$$

and for $j \in \{1, \dots, d_{\ell+1}-1\}$,

$$\frac{s_j\Delta_\ell}{2} = b\mathbb{1}(j \text{ odd}) + a_{\ell+1}\mathbb{1}(j \text{ even}). \tag{25}$$

Using arguments as for the rewriting of $\sum_{j=1}^{d_{\ell+1}-1} r_j\, z_{\ell+1}^{(j)}(x)$ above,

$$\begin{aligned}
\sum_{j=0}^{d_{\ell+1}-1} s_j\, z_{\ell+1}^{(j)}(x) &= \frac{2s_0}{D_\ell} \sum_{k=1}^{D_\ell-1} \Big(x - \frac{k}{D_\ell}\Big)_+ + \Big(\sum_{j=1}^{d_{\ell+1}-1} s_j\Big) \sum_{k=1}^{D_\ell-1} b\Big(x - \frac{k}{D_\ell}\Big)_+ \\
&\quad + \sum_{k=0}^{D_\ell-1} \sum_{j=1}^{d_{\ell+1}-1} \frac{s_j \Delta_\ell}{2} \Big(\mathbb{1}(k \text{ even}) \Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ + \mathbb{1}(k \text{ odd}) \Big(x - \frac{k}{D_\ell} - \frac{d_{\ell+1}-j}{D_{\ell+1}}\Big)_+ \Big) \\
&= \sum_{k=1}^{D_\ell-1} a_{\ell+1} \Big(x - \frac{k}{D_\ell}\Big)_+ + \sum_{k=0}^{D_\ell-1} \sum_{j=1}^{d_{\ell+1}-1} \frac{s_j \Delta_\ell}{2} \Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ \\
&= \sum_{k=1}^{D_\ell-1} a_{\ell+1} \Big(x - \frac{k}{D_\ell}\Big)_+ + \sum_{k=0}^{D_\ell-1} \sum_{j=1}^{d_{\ell+1}-1} \Big(b\mathbb{1}(j \text{ odd}) + a_{\ell+1}\mathbb{1}(j \text{ even}) \Big) \Big(x - \frac{k}{D_\ell} - \frac{j}{D_{\ell+1}}\Big)_+ \\
&= \sum_{q=1}^{D_{\ell+1}-1} \Big(b\mathbb{1}(q \text{ odd}) + a_{\ell+1}\mathbb{1}(q \text{ even}) \Big) \Big(x - \frac{q}{D_{\ell+1}}\Big)_+ \\
&=: u_{\ell+1}^{(2)}(x)
\end{aligned}$$

where we used (24) and (25) for the second and third equalities, respectively. Hence, $u_{\ell+1}^{(2)}(x)$ is also a non-negative affine combination of $(z_{\ell+1}^{(j)}(x))_{j=0}^{d_{\ell+1}-1}$. Finally, we get the remaining assertion by noticing that

$$u_{\ell+1}^{(0)}(x) = \sum_{k=1}^{D_{\ell+1}-1} \frac{2}{D_{\ell+1}} \Big(x - \frac{k}{D_{\ell+1}}\Big)_+ = \frac{2}{D_{\ell+1}(a_{\ell+1}+b)} \Big(u_{\ell+1}^{(1)}(x) + u_{\ell+1}^{(2)}(x)\Big)$$

can also be represented as a non-negative affine combination of $(z_{\ell+1}^{(j)}(x))_{j=0}^{d_{\ell+1}-1}$. □

We also present an alternative theorem and proof below, which is more accessible. The network construction below is similar to that of the original proof for ReLU networks (e.g., Yarotsky (2017)), and highlights the key idea behind the exponential approximation rate.

Lemma B.12. *Let $\psi : [0,1] \to [0,1]$ be the function $\psi(u) = |2u-1|$ and define $\psi^{\circ k} = \psi \circ \ldots \circ \psi$ as the k-fold composition of ψ. Then,*

(i) $\psi^{\circ k} + (1-\psi)^{\circ k} = 1$,

(ii) for a real number u denote by $\{u\} := u - \lfloor u \rfloor \in [0,1)$ the fractional part. For any $y \in [0,1]$, and any $k = 1, 2, \ldots,$

$$\psi^{\circ k}(y) = \psi\big(\{2^{k-1}y\}\big),$$

(iii) for any $L = 1, 2, \ldots,$ the function

$$x \mapsto \sum_{k=1}^{L} 4^{-k}\big(1 - \psi^{\circ k}(x)\big)$$

interpolates $x \mapsto x(1-x)$ at the point $\ell 2^{-L}$ for any $\ell = 0, \ldots, 2^L$, and is affine on the interval $[\ell 2^{-L}, (\ell+1)2^{-L}]$ for any $\ell = 0, \ldots, 2^L - 1$,

(iv) for any $L = 1, 2, \ldots$

$$\sup_{x \in [0,1]} \Big| x + \sum_{k=1}^{L} 4^{-k}\big(\psi^{\circ k}(x) - 1\big) - 2^{-2L-3} - x^2 \Big| = 2^{-2L-3}.$$

Before we give the proof of the lemma, we state the main result.

Theorem B.13 (Special case of Lemma B.7)**.** *For any $L \in \mathbb{N}$, there exists a HyCNN $h : \mathbb{R} \to \mathbb{R}$ with L hidden layers and 2 neurons per layer, such that*

$$\int_0^1 |h(x) - x^2|\, dx \le \sup_{x \in [0,1]} |h(x) - x^2| = 2^{-2L-3}.$$

Proof. For real numbers u, v, we have $2\max(u,v) = u + v + |u - v|$. Therefore, a layer of the form

$$\mathbf{z} \mapsto \max\left(V^1\mathbf{z} + W^1\mathbf{x} + \mathbf{b}^1, V^2\mathbf{z} + W^2\mathbf{x} + \mathbf{b}^2\right)$$

with V^1, V^2 entrywise non-negative, is equivalent to the existence of matrices A, B, D, E and vectors $\mathbf{a}, \mathbf{b}$ such that the absolute value of every entry in D is upper bounded by the corresponding entry of A (implying that $A - D$ and $A + D$ are entrywise non-negative) and

$$\mathbf{z} \mapsto A\mathbf{z} + B\mathbf{x} + \mathbf{a} + \left|D\mathbf{z} + E\mathbf{x} + \mathbf{b}\right|, \tag{26}$$

where $|\cdot|$ is the componentwise absolute value. Let $\psi : [0,1] \to [0,1]$ be the function $\psi(u) = |2u - 1|$ and define $\psi^{\circ k} = \psi \circ \ldots \circ \psi$ as the k-fold composition of ψ. For any $m = 1, 2, \ldots$ we now show that there exists a network with L hidden layers, one input neuron, one output neuron, and two neurons in each of the hidden layers, such that the m-th hidden layer computes

$$\begin{pmatrix} 2^{-2m}\psi^{\circ m}(x) + \sum_{k=1}^{m} 2^{-2k}\psi^{\circ k}(x) \\ -2^{-2m}\psi^{\circ m}(x) + \sum_{k=1}^{m} 2^{-2k}\psi^{\circ k}(x) \end{pmatrix} = \begin{pmatrix} 2^{-2m}\psi^{\circ m}(x) + \sum_{k=1}^{m} 2^{-2k}\psi^{\circ k}(x) \\ \sum_{k=1}^{m-1} 2^{-2k}\psi^{\circ k}(x) \end{pmatrix}.$$

Then, we can average the two activations in the output layer and add the term $x - \sum_{k=1}^{L} 2^{-2k} - 2^{-2L-3}$ to get a network computing

$$x + \sum_{k=1}^{L} 2^{-2k}\left(\psi^{\circ k}(x) - 1\right) - 2^{-2L-3},$$

and get the assertion by applying Lemma B.12 (iv).

The case $m = 1$ follows directly from the form (26) with $A = B = \mathbf{a} = D = (0,0)^\top$, $E = (1/2, 0)^\top$, $\mathbf{b} = (-1/4, 0)^\top$. For the induction step from m to $m+1$, take $B = E = \mathbf{a} = (0,0)^\top$,

$$A = \begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}, \quad D = \begin{pmatrix} 1/2 & -1/2 \\ 0 & 0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} -2^{-2m-1} \\ 0 \end{pmatrix}.$$

The choice satisfies the constraint that the entries in D are in absolute value dominated by the corresponding entries in A. Observe that

$$2^{-2m-1}\psi^{\circ(m+1)}(x) = 2^{-2m-1}|2\psi^{\circ m}(x) - 1| = |2^{-2m}\psi^{\circ m}(x) - 2^{-2m-1}|.$$

Let $\mathbf{z}_m$ denote the activations of the previous layer. Using the induction hypothesis, $D\mathbf{z}_m = (2^{-2m}\psi^{\circ m}(x), 0)^\top$ and

$$A\mathbf{z}_m + |D\mathbf{z}_m + \mathbf{b}| = \sum_{k=1}^{m} 2^{-2k}\psi^{\circ k}(x)\begin{pmatrix} 1 \\ 1 \end{pmatrix} + 2^{-2m-2}\psi^{\circ(m+1)}(x)\begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

proving the induction hypothesis.

□

Proof of Lemma B.12. (i): We prove $\psi^{\circ k} + (1-\psi)^{\circ k} = 1$ using induction on k. The formula holds for $k = 1$. To go from k to $k+1$, observe that by $\psi^{\circ(k+1)} = \psi^{\circ k} \circ \psi$, $(1-\psi)^{\circ(k+1)} = (1-\psi)^{\circ k} \circ (1-\psi)$, the induction hypothesis, and the fact that ψ is symmetric around $1/2$ in the sense that $\psi(u) = \psi(1-u)$, thus, $\psi \circ \psi = \psi \circ (1-\psi)$, we obtain $\psi^{\circ(k+1)} + (1-\psi)^{\circ(k+1)} = \psi^{\circ k} \circ \psi + 1 - \psi^{\circ k} \circ (1-\psi) = 1$, proving the induction step.

(ii): The formula holds trivially for $k = 1$.

We now prove it for $k = 2$: Note that $y = \{2y\}/2$ if $0 \le y < 1/2$ and $y = 1/2 + \{2y\}/2$ if $1/2 \le y < 1$. Therefore, $\psi(y) = |2y - 1| = |\{2y\} - 1| = 1 - \{2y\}$ if $0 \le y < 1/2$ and $\psi(y) = |2y - 1| = \{2y\}$ if $1/2 \le y < 1$. Since $\psi(u) = \psi(1-u)$ for all real numbers u, we find $\psi \circ \psi(y) = \psi(\{2y\})$ for all $0 \le y < 1$. For $y = 1$, the formula holds since $\psi(0) = \psi(1) = 1$.

To prove $\psi^{\circ k}(y) = \psi(\{2^{k-1}y\})$ for general k, it is enough to show the induction step $k \to k+1$. Using the induction hypothesis and the case $k = 2$ proved above, $\psi^{\circ(k+1)}(y) = \psi(\psi^{\circ k}(y)) = \psi \circ \psi(\{2^{k-1}y\}) = \psi(\{2\{2^{k-1}y\}\}) = \psi(\{2^k y\})$.

(iii): By (ii), $\psi^{\circ k}(y) = \psi(\{2^{k-1}y\})$. Since $\psi(x) = |2x-1|$ is affine on $[0,1/2]$ and $[1/2,1]$, it follows that $\psi^{\circ k}$ is piecewise-affine on the intervals $[\ell/2^k, (\ell+1)/2^k]$ with endpoints $\psi^{\circ k}(\ell/2^k) = \psi(1/2) = 0$ if ℓ is odd and $\psi^{\circ k}(\ell/2^k) = \psi(0) = 1$ if ℓ is even.

For convenience, write $g(x) = x(1-x)$. In the next step, we show that for any $L \geq 1$ and any $\ell = 0, 1, \ldots, 2^L$,

$$g(\ell 2^{-L}) = \sum_{k=1}^{L} 4^{-k}\big(1 - \psi^{\circ k}(\ell 2^{-L})\big).$$

We prove this by induction over L. For $L = 1$ the result can be checked directly. For the inductive step, suppose that the claim holds for L. If ℓ is even we use that $\psi^{\circ(L+1)}(\ell 2^{-L-1}) = 1$ to obtain that $g(\ell 2^{-L-1}) = \sum_{k=1}^{L} 4^{-k}(1-\psi^{\circ k}(\ell 2^{-L-1})) = \sum_{k=1}^{L+1} 4^{-k}(1-\psi^{\circ k}(\ell 2^{-L-1}))$. It thus remains to consider ℓ odd. Recall that $x \mapsto \sum_{k=1}^{L} 4^{-k}(1-\psi^{\circ k}(x))$ is affine on $[(\ell-1)2^{-L-1}, (\ell+1)2^{-L-1}]$ and observe that for any real t,

$$g(x) = t^2 + \frac{g(x-t) + g(x+t)}{2}.$$

For odd ℓ, choose $x = \ell 2^{-L-1}$ and $t = 2^{-L-1}$. Applying the identity and $\psi^{\circ(L+1)}(\ell 2^{-L-1}) = 0$ yields

$$\begin{aligned}
g(\ell 2^{-L-1}) &= 2^{-2L-2} + \frac{g\big((\ell-1)2^{-L-1}\big) + g\big((\ell+1)2^{-L-1}\big)}{2} \\
&= 4^{-L-1} + \frac{1}{2}\Bigg(\sum_{k=1}^{L} 4^{-k}\Big(1 - \psi^{\circ k}\big((\ell-1)2^{-L-1}\big)\Big) + \sum_{k=1}^{L} 4^{-k}\Big(1 - \psi^{\circ k}\big((\ell+1)2^{-L-1}\big)\Big)\Bigg) \\
&= 4^{-L-1} + \sum_{k=1}^{L} 4^{-k}\big(1 - \psi^{\circ k}(\ell 2^{-L-1})\big) \\
&= \sum_{k=1}^{L+1} 4^{-k}\big(1 - \psi^{\circ k}(\ell 2^{-L-1})\big).
\end{aligned}$$

This completes the inductive step and proves assertion *(iii)*.

(iv): For a quadratic function h with $h(a) = h(b) = 0$ and $K = h''(a) = h''(b)$, one has that $h(x) = \frac{K}{2}(x-a)(x-b)$. Since $g''(x) = -2$, for any $\ell = 0, \ldots, 2^L - 1$, and any $x \in [\ell 2^{-L}, (\ell+1)2^{-L}]$,

$$g(x) - \sum_{k=1}^{L} 4^{-k}\big(1 - \psi^{\circ k}(x)\big) = -\big(x - (\ell+1)2^{-L}\big)\big(x - \ell 2^{-L}\big) = \big((\ell+1)2^{-L} - x\big)\big(x - \ell 2^{-L}\big). \quad (27)$$

For any $x \in [\ell 2^{-L}, (\ell+1)2^{-L}]$,

$$0 \leq \Big(\frac{\ell+1}{2^L} - x\Big)\Big(x - \frac{\ell}{2^L}\Big) \leq \Big(\frac{\ell+1}{2^L} - \frac{\ell+1/2}{2^L}\Big)\Big(\frac{\ell+1/2}{2^L} - \frac{\ell}{2^L}\Big) = 2^{-2L-2}.$$

The second inequality becomes an equality for the interval midpoint $x = (\ell+1/2)2^{-L}$. This shows that

$$\sup_{x\in[0,1]} \Big|g(x) - \sum_{k=1}^{L} 4^{-k}\big(1 - \psi^{\circ k}(x)\big) - 2^{-2L-3}\Big| = 2^{-2L-3}.$$

The assertion follows using that $g(x) = x(1-x) = x - x^2$.

□

Proof of Theorem 3.3. By Lemma B.7, there exists a HyCNN $h\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L hidden layers, and $(d_1, \ldots, d_L)$ neurons per layer such that

$$\sup_{x\in[0,1]} |h(x) - x^2| \leq \frac{1}{8\prod_{\ell=1}^{L} d_\ell^2}.$$

By Lemma B.2, there exists a HyCNN $g\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$ and the same architecture as h such that

$$\sup_{x\in[0,1]} |g(x) - h(x)| \leq \Big(\frac{1}{7.99} - \frac{1}{8}\Big)\frac{1}{\prod_{\ell=1}^{L} d_\ell^2}.$$

The assertion follows from the triangle inequality. □

B.4 Proofs for monomial functions

In this section, we provide the proof of Theorem 3.5. We first establish the following lemma, which will be used to prove the main result. The proof of Theorem 3.5 then follows by applying Lemma B.2.

Lemma B.14 (Monomial Function Approximation with maxout unit)**.** *For any integer $n \geq 2$, $L \in \mathbb{N}$ and odd number $m \geq 3$, there exists a HyCNN $h\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, $\lceil \log_2(n) \rceil L$ layers and m neurons per layer such that*

$$\sup_{x \in [0,1]} \left| h(x) - x^n \right| \leq \frac{n}{2}(m-1)^{-2L}.$$

The following lemma states that how HyCNNs can approximate the functional operators $\mathcal{D}_0$ and $\mathcal{D}_1$, defined by

$$\mathcal{D}_0[f](x) := f^2(x), \qquad \mathcal{D}_1[f](x) := f^2(x)/x.$$

These operators are crucially used in the proof of Lemma B.14. Approximating $\mathcal{D}_0$ is straightforward from Lemma B.7, whereas approximating $\mathcal{D}_1$ is a non-trivial problem.

Lemma B.15. *Consider a HyCNN $g\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L' hidden layers and m' neurons per hidden layer. Assume that $0 \leq g(x) \leq x$ for every $x \in [0,1]$. Then, for any $L \in \mathbb{N}$ and $m \in 2\mathbb{N}$,*

(i) there exists a HyCNN $\widetilde{g}_0\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, $L' + L$ hidden layers and $\max(m', m+1)$ neurons per hidden layer such that $0 \leq \widetilde{g}_0(x) \leq x$ for every $x \in [0,1]$ and

$$\sup_{x \in [0,1]} \left| \widetilde{g}_0(x) - \mathcal{D}_0[g](x) \right| \leq \frac{1}{4m^{2L}}.$$

(ii) there exists a HyCNN $\widetilde{g}_1\colon \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, $L' + L$ hidden layers and $\max(m', m+1)$ neurons per hidden layer such that $0 \leq \widetilde{g}_1(x) \leq x$ for every $x \in [0,1]$ and

$$\sup_{x \in [0,1]} \left| \widetilde{g}_1(x) - \mathcal{D}_1[g](x) \right| \leq \frac{1}{4m^{2L}}.$$

To prove this lemma, we need the following two key results. First, we modify our network to approximate the quadratic function, in order to ensure that the *skip-connection weight matrices*, i.e., the matrices $\{W_\ell^1, W_\ell^2\}_{\ell=0}^{L-1}$ and W_L in (2), are element-wise non-negative. This is relevant for proving Lemma B.15(i), concretely in order to ensure composability of HyCNNs according to Lemma B.4.

Lemma B.16. *For any $L \in \mathbb{N}$ and $m \in 2\mathbb{N}$, there exists a HyCNN $f : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L hidden layers and m neurons per hidden layer such that*

$$\sup_{x \in [0,1]} \left| f(x) - x^2 \right| \leq \frac{1}{4m^{2L}}$$

and $0 \leq f(x) \leq x$ for every $x \in [0,1]$. Moreover, all the skip-connection weight matrices of f are element-wise non-negative.

Proof. We define $M := m^L$. By adding $(8M^2)^{-1}$ to (16) in the proof of Lemma B.7, we can construct a HyCNN $f : \mathbb{R} \to \mathbb{R}$ with L hidden layers and m neurons per hidden layer such that

$$f(x) = \frac{x}{M} + \frac{2}{M} \sum_{k=1}^{M-1} \left(x - \frac{k}{M} \right)_+,$$

and all the skip-connection weight matrices of f are element-wise non-negative (see Remark B.11). By Lemma B.8, we obtain

$$\sup_{x \in [0,1]} |f(x) - x^2| \leq (8M^2)^{-1} + \sup_{x \in [0,1]} \left| -(8M^2)^{-1} + f(x) - x^2 \right| \leq (4M^2)^{-1} = \frac{1}{4m^{2L}}.$$

Notice that $f(0) = 0$ and

$$f(1) = \frac{1}{M} + \frac{2(M-1)}{M} - \frac{M(M-1)}{M^2} = 1.$$

Since f is non-decreasing and convex, we obtain the remaining assertion by

$$0 \le f(x) \le (1-x)f(0) + xf(1) = x.$$

□

The next lemma serves as a crucial tool to prove Lemma B.15(ii).

Lemma B.17. *Consider a HyCNN $h\colon \mathbb{R}^d \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L hidden layers and m neurons per hidden layer, specified for $\ell \in \{0, \dots, L-1\}$ by*

$$h(\mathbf{x}) = \mathbf{v}_L^\top \mathbf{z}_L(\mathbf{x}) + \mathbf{w}_L^\top \mathbf{x} + b_L, \quad \mathbf{z}_{\ell+1}(\mathbf{x}) = \max_{k\in\{1,2\}} \left(V_\ell^k \mathbf{z}_\ell(\mathbf{x}) + W_\ell^k \mathbf{x} + \mathbf{b}_\ell^k\right), \quad \mathbf{z}_0(\mathbf{x}) = 0.$$

For $\mathbf{x}_1 \in \mathbb{R}^d$ and $x_2 \in \mathbb{R}$, define $\widetilde{h}(\mathbf{x}_1, x_2)$ by

$$\begin{aligned} \widetilde{h}(\mathbf{x}_1, x_2) &:= \mathbf{v}_L^\top \widetilde{\mathbf{z}}_L(\mathbf{x}_1, x_2) + \mathbf{w}_L^\top \mathbf{x}_1 + b_L x_2, \\ \widetilde{\mathbf{z}}_{\ell+1}(\mathbf{x}_1, x_2) &:= \max_{k\in\{1,2\}} \left(V_\ell^k \widetilde{\mathbf{z}}_\ell(\mathbf{x}_1, x_2) + W_\ell^k \mathbf{x}_1 + \mathbf{b}_\ell^k x_2\right). \end{aligned} \tag{28}$$

with $\mathbf{z}_0(\mathbf{x}_1, x_2) = \mathbf{0}_{d+1}$. Then, it follows that $\widetilde{h}(\mathbf{u}y, y) = yh(\mathbf{u})$ for any $\mathbf{u} \in \mathbb{R}^d$ and $y \ge 0$.

Proof. The claim follows from induction on the depth ℓ. For $\ell = 0$, we have

$$\begin{aligned} \widetilde{\mathbf{z}}_1(\mathbf{u}y, y) &= \max_{k\in\{1,2\}} \left(W_0^k \mathbf{u}y + \mathbf{b}_0^k y\right) \\ &= y \max_{k\in\{1,2\}} \left(W_0^k \mathbf{u} + \mathbf{b}_0^k\right) \\ &= y\mathbf{z}_1(\mathbf{u}). \end{aligned}$$

Assuming that $\widetilde{\mathbf{z}}_\ell(\mathbf{u}y, y) = y\mathbf{z}_\ell(\mathbf{u})$ for some $\ell \in \{0, \dots, L-1\}$, we get

$$\begin{aligned} \widetilde{\mathbf{z}}_{\ell+1}(\mathbf{u}y, y) &= \max_{k\in\{1,2\}} \left(V_\ell^k \widetilde{\mathbf{z}}_\ell(\mathbf{u}y, y) + W_\ell^k \mathbf{u}y + \mathbf{b}_\ell^k y\right) \\ &= \max_{k\in\{1,2\}} \left(yV_\ell^k \mathbf{z}_\ell(\mathbf{u}) + W_\ell^k \mathbf{u}y + \mathbf{b}_\ell^k y\right) \\ &= y \max_{k\in\{1,2\}} \left(V_\ell^k \mathbf{z}_\ell(\mathbf{u}) + W_\ell^k \mathbf{u} + \mathbf{b}_\ell^k\right) \\ &= y\mathbf{z}_{\ell+1}(\mathbf{u}), \end{aligned}$$

which proves the induction step. Finally,

$$\begin{aligned} \widetilde{h}(\mathbf{u}y, y) &= \mathbf{v}_L^\top \widetilde{\mathbf{z}}_L(uy, y) + \mathbf{w}_L^\top \mathbf{u}y + b_L y \\ &= y\mathbf{v}_L^\top \mathbf{z}_L(\mathbf{u}) + \mathbf{w}_L^\top \mathbf{u}y + b_L y \\ &= yh(\mathbf{u}). \end{aligned}$$

□

Proof of Lemma B.15. (i). By Lemma B.16, we can construct a HyCNN $h : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L hidden layers and m neurons per hidden layer such that

$$\sup_{x\in[0,1]} |h(x) - x^2| \le \frac{1}{4m^{2L}},$$

$0 \le h(x) \le x$ for every $x \in [0, 1]$, and all the skip-connection weight matrices of h are element-wise non-negative. By Lemma B.4, there exists a HyCNN $\widetilde{g}_0 : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, $L' + L$ hidden layers and $\max(m', m+1)$ neurons per hidden layer such that $\widetilde{g}_0(x) = h \circ g(x)$. We obtain the upper bound of the approximation error by

$$\begin{aligned} \sup_{x\in[0,1]} \left|\widetilde{g}_0(x) - g^2(x)\right| = \sup_{x\in[0,1]} \left|h \circ g(x) - g^2(x)\right| &\le \sup_{z\in[0,1]} \left|h(z) - z^2\right| \\ &\le \frac{1}{4m^{2L}}, \end{aligned}$$

provided that $0 \le g(x) \le 1$. Finally, from $0 \le h \circ g(x) \le x$ for every $x \in [0, 1]$, we get $0 \le \widetilde{g}_0(x) \le x$.

(ii). By Lemma B.16, we can construct a HyCNN $h : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L hidden layers and m neurons per hidden layer such that

$$\sup_{x\in[0,1]} |h(x) - x^2| \le \frac{1}{4m^{2L}},$$

$0 \le h(x) \le x$ for every $x \in [0, 1]$, and all the skip-connection weight matrices of h are element-wise non-negative.

By applying Lemma B.17 to the HyCNN h, we define a function $\widetilde{h}: \mathbb{R}^2 \to \mathbb{R}$ by

$$\widetilde{h}(x_1, x_2) := \mathbf{v}_L^\top \widetilde{\mathbf{z}}_L(x_1, x_2) + w_L x_1 + b_L x_2,$$
$$\widetilde{\mathbf{z}}_{\ell+1}(x_1, x_2) := \max_{k\in\{1,2\}} \left(V_\ell^k \widetilde{\mathbf{z}}_\ell(x_1, x_2) + \mathbf{w}_\ell^k x_1 + \mathbf{b}_\ell^k x_2\right)$$

with $\widetilde{z}_0(x_1, x_2) = 0$, where the parameters $\mathbf{v}_L, w_L, \mathbf{b}_L$, $\{V_\ell^1, \mathbf{w}_\ell^1, \mathbf{b}_\ell^1\}_{\ell=0}^{L-1}$ and $\{V_\ell^2, \mathbf{w}_\ell^2, \mathbf{b}_\ell^2\}_{\ell=0}^{L-1}$ coincide with those of the HyCNN h. Then, $\widetilde{h}$ satisfies $\widetilde{h}(uy, y) = yh(u)$ for any $u \in \mathbb{R}$ and $y \ge 0$. By plugging in $u = g(x)/x$ and $y = x$, we get

$$\widetilde{h}\big(g(x), x\big) = xh\big(g(x)/x\big), \qquad \forall x \in [0, 1].$$

All the $\mathbf{v}_L, w_L$ and $\{V_\ell^1, \mathbf{w}_\ell^1, V_\ell^2, \mathbf{w}_\ell^2\}_{\ell=0}^{L-1}$ of h (and of $\widetilde{h}$) are element-wise non-negative. Therefore, we can apply Lemma B.4 to obtain a HyCNN $\widetilde{g} : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, $L' + L$ hidden layers and $\max(m', m + 1)$ neurons per hidden layer such that $\widetilde{g}(x) = \widetilde{h}(g(x), x)$. We obtain the upper bound of the approximation error by

$$\begin{aligned}\sup_{x\in[0,1]} \left|\widetilde{g}(x) - g^2(x)/x\right| = \sup_{x\in[0,1]} \left|xh\big(g(x)/x\big) - x(g(x)/x)^2\right| &\le \sup_{x\in[0,1]} \left|h\big(g(x)/x\big) - (g(x)/x)^2\right| \\ &\le \frac{1}{4m^{2L}},\end{aligned}$$

provided that $0 \le g(x)/x \le 1$. Finally, from $0 \le h(g(x)/x) \le 1$ for every $x \in [0, 1]$, we get $0 \le \widetilde{g}(x) \le x$. □

Proof of Lemma B.14. For functions $f: \mathbb{R} \to \mathbb{R}$, define the operators

$$\mathcal{D}_0[f](x) := f^2(x), \qquad \mathcal{D}_1[f](x) := f^2(x)/x.$$

Further, let $\phi_m(x) = x^m$ for $m \in \mathbb{N}$. We define $k := \lceil \log_2(n) \rceil$ and represent $2^k - n$ through its binary digits. Since $0 \le 2^k - n \le 2^{k-1} - 1$, there exist $b_1, \ldots, b_{k-1}$ with $b_i \in \{0, 1\}$ for $i = 1, \ldots, k - 1$ such that

$$2^k - n = \sum_{i=1}^{k-1} b_i 2^{i-1}.$$

Set $b_k := 0$. Since $\mathcal{D}_b[\phi_m](x) = \phi_{2m-b}(x)$ for any $b \in \{0, 1\}$ and $m \in \mathbb{N}$, we get

$$\begin{aligned}\mathcal{D}_{b_1} \circ \mathcal{D}_{b_2} \circ \ldots \circ \mathcal{D}_{b_{k-2}} \circ \mathcal{D}_{b_{k-1}} \circ \mathcal{D}_{b_k}[\phi_1](x) &= \mathcal{D}_{b_1} \circ \mathcal{D}_{b_2} \circ \ldots \circ \mathcal{D}_{b_{k-2}} \circ \mathcal{D}_{b_{k-1}}[\phi_2](x) \\ &= \phi_n(x) \end{aligned} \tag{29}$$

since

$$\begin{aligned}\overbrace{2\Big(\ldots 2\big(2(2\cdot 2 - b_{k-1}) - b_{k-2}\big) - b_{k-3} \ldots\Big)}^{k-1 \text{ times}} - b_1 &= 2^k - 2^{k-2} b_{k-1} - 2^{k-3} b_{k-2} - \ldots - b_1 \\ &= 2^k - \sum_{i=1}^{k-1} b_i 2^{i-1} = n.\end{aligned}$$

We approximate the composition with HyCNNs and track the approximation error using backward induction, starting with k and then going down to 1. By Lemma B.16, there exists a HyCNN $h_1 : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1, a_2) = \max(a_1, a_2)$, L hidden layers and m neurons per hidden layer such that

$$\sup_{x\in[0,1]} \left|h_1(x) - \mathcal{D}_{b_k}[\phi_1](x)\right| = \sup_{x\in[0,1]} \left|h_1(x) - x^2\right| \le \frac{1}{4m^{2L}}$$

and $0 \le h_1(x) \le x$ for every $x \in [0,1]$. Assume that for $i \in \{1,2,\dots,k-1\}$, there exists a HyCNN $h_i : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1,a_2) = \max(a_1,a_2)$, Li hidden layers and $m+1$ neurons per hidden layer such that

$$\sup_{x\in[0,1]} \left|h_i(x) - \underbrace{\mathcal{D}_{b_{k+1-i}} \circ \mathcal{D}_{b_{k+2-i}} \circ \ldots \circ \mathcal{D}_{b_k}}_{i \text{ times}}[\phi_1](x)\right| \le \frac{2^i-1}{4m^{2L}}$$

and $0 \le h_i(x) \le x$ for every $x \in [0,1]$. For $x \in [0,1]$, we also have

$$0 \le \mathcal{D}_{b_{k+1-i}} \circ \ldots \circ \mathcal{D}_{b_k}[\phi_1](x) \le x.$$

By applying Lemma B.15-(i) for the case $b_{k-i} = 0$ and Lemma B.15-(ii) for the case $b_{k-i} = 1$, there exists a HyCNN $h_{i+1} : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1,a_2) = \max(a_1,a_2)$, $L(i+1)$ hidden layers and $m+1$ neurons per hidden layer such that

$$\sup_{x\in[0,1]} \left|h_{i+1}(x) - \mathcal{D}_{b_{k-i}}[h_i](x)\right| \le \frac{1}{4m^{2L}}$$

and $0 \le h_{i+1}(x) \le x$ for every $x \in [0,1]$. Using the triangle inequality and the induction hypothesis,

$$\begin{aligned}
&\sup_{x\in[0,1]} \left|h_{i+1}(x) - \underbrace{\mathcal{D}_{b_{k-i}} \circ \mathcal{D}_{b_{k+1-i}} \circ \ldots \circ \mathcal{D}_{b_k}}_{(i+1) \text{ times}}[\phi_1](x)\right| \\
&\le \sup_{x\in[0,1]} \left|h_{i+1}(x) - \mathcal{D}_{b_{k-i}}[h_i](x)\right| + \sup_{x\in[0,1]} \left|\mathcal{D}_{b_{k-i}}[h_i](x) - \mathcal{D}_{b_{k-i}} \circ \mathcal{D}_{b_{k+1-i}} \circ \ldots \circ \mathcal{D}_{b_k}[\phi_1](x)\right| \\
&= \sup_{x\in[0,1]} \left|h_{i+1}(x) - \mathcal{D}_{b_{k-i}}[h_i](x)\right| + \sup_{x\in[0,1]} \left|\frac{h_i^2(x)}{x^{b_{k-i}}} - \frac{\left(\mathcal{D}_{b_{k+1-i}} \circ \ldots \circ \mathcal{D}_{b_k}[\phi_1](x)\right)^2}{x^{b_{k-i}}}\right| \\
&\le \frac{1}{4m^{2L}} + \sup_{x\in[0,1]} \frac{\left|\left(h_i(x) - \mathcal{D}_{b_{k+1-i}} \circ \ldots \circ \mathcal{D}_{b_k}[\phi_1](x)\right)\left(h_i(x) + \mathcal{D}_{b_{k+1-i}} \circ \ldots \circ \mathcal{D}_{b_k}[\phi_1](x)\right)\right|}{x^{b_{k-i}}} \\
&\le \frac{1}{4m^{2L}} + 2 \cdot \frac{2^i-1}{4m^{2L}} = \frac{2^{i+1}-1}{4m^{2L}}.
\end{aligned}$$

Hence, by backward induction on i, there exists a HyCNN $h_k : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1,a_2) = \max(a_1,a_2)$, Lk hidden layers and $m+1$ neurons per hidden layer such that

$$\sup_{x\in[0,1]} \left|h_k(x) - \mathcal{D}_{b_1} \circ \mathcal{D}_{b_2} \circ \ldots \circ \mathcal{D}_{b_k}[\phi_1](x)\right| \le \frac{2^k-1}{4m^{2L}} < \frac{2^{k-2}}{m^{2L}} \le \frac{n}{2m^{2L}}.$$

The assertion then follows by recalling (29) and replacing m by $m-1$. □

Proof of Theorem 3.5. By Lemma B.14, there exists a HyCNN $h : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1,a_2) = \max(a_1,a_2)$, $\lceil \log_2(n) \rceil L$ layers and m neurons per layer such that

$$\sup_{x\in[0,1]} |h(x) - x^2| \le \frac{n}{2}(m-1)^{-2L}.$$

By Lemma B.2, there exists a HyCNN $g : \mathbb{R} \to \mathbb{R}$ with $\overline{\sigma}(a_1,a_2) = \overline{\sigma}_\tau(a_1,a_2)$ and the same architecture as h such that

$$\sup_{x\in[0,1]} |g(x) - h(x)| \le \left(\frac{1}{1.99} - \frac{1}{2}\right)(m-1)^{-2L}.$$

The assertion follows from the triangle inequality. □

B.5 Proof of Inclusions (5)

Proof. We write $\mathbf{x} = (x_1, \dots, x_d)^\top$. Theorems 3.2 and 3.3 yield

$$\inf_{f \in \mathrm{ICNN}_d(L,m)} \sup_{\mathbf{x} \in [0,1]^d} |f(\mathbf{x}) - x_1^2| \geq \frac{1}{8(m+2(L-1)m)^2} > \frac{1}{8(2mL)^2}$$

and

$$\inf_{h \in \mathrm{HyCNN}_d(L',m')} \sup_{\mathbf{x} \in [0,1]^d} |h(\mathbf{x}) - x_1^2| \leq \frac{1}{8(m')^{2L'}} \leq \frac{1}{8(2mL)^2}.$$

respectively. Hence, we get $\mathrm{HyCNN}_d(L', m') \subsetneq \mathrm{ICNN}_d(L, m)$.

The second assertion follows by setting $V_\ell^2 = W_\ell^2 = 0$ and $\mathbf{b}_\ell^2 = 0$ for each $\ell = 0, \dots, L-1$ in (2).

To establish the last inclusion, write $(x_1, \dots, x_d)_+ := (x_1 \vee 0, \dots, x_d \vee 0)$. For a given HyCNN $h \in \mathrm{HyCNN}_d(L, m)$, we construct a ReLU network $f \in \mathrm{ReLU}_d(L, 3m + 2d)$ that represents h. In each hidden layer of f, we store $\mathbf{x}_+$ and $(-\mathbf{x})_+$, which represent the positive and negative parts of the input vector, respectively. This requires $2d$ neurons per layer, and these neurons can be used to represent skip connections from the input via

$$\mathbf{a}^\top \mathbf{x} = \mathbf{a}^\top \big(\mathbf{x}_+ - (-\mathbf{x})_+\big) = \mathbf{a}^\top \mathbf{x}_+ - \mathbf{a}^\top (-\mathbf{x})_+.$$

On the other hand, from $x \vee y = x + (y - x)_+$ and $x = x_+ - (-x)_+$, we have

$$\begin{aligned}(\mathbf{a}_1^\top \mathbf{z} + b_1) \vee (\mathbf{a}_2^\top \mathbf{z} + b_2) &= (\mathbf{a}_1^\top \mathbf{z} + b_1) + \big((\mathbf{a}_2 - \mathbf{a}_1)^\top \mathbf{z} + b_2 - b_1\big)_+ \\ &= (\mathbf{a}_1^\top \mathbf{z} + b_1)_+ - (-\mathbf{a}_1^\top \mathbf{z} - b_1)_+ + \big((\mathbf{a}_2 - \mathbf{a}_1)^\top \mathbf{z} + b_2 - b_1\big)_+.\end{aligned}$$

Hence, instead of m neurons in each hidden layer of h computing the left-hand side of the previous identity, $3m$ neurons in the ReLU network f can compute the right-hand side. This proves the assertion. □

B.6 Proofs for additional theoretical results

Proof of Theorem A.1. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a multivariate (standard or leaky) ReLU ICNN as in the claim. Then, by continuity of f and $\|\cdot\|_2^2$, there exists $\mathbf{x}' \in [0,1]^{d-1}$ such that

$$\int_{[0,1]^d} \big|f(\mathbf{x}) - \|\mathbf{x}\|_2^2\big| \, d\mathbf{x} \geq \int_0^1 \big|f(x, \mathbf{x}') - x^2 - \|\mathbf{x}'\|_2^2\big| \, dx$$

In particular, the function $x \mapsto f(x, \mathbf{x}')$ is a univariate (standard or leaky) ReLU ICNN with L hidden layers and d_ℓ neurons in layer ℓ. Hence, by Theorem 3.2, we have

$$\int_0^1 \big|f(x, \mathbf{x}') - x^2 - \|\mathbf{x}'\|_2^2\big| \, dx \geq \frac{1}{16(d_1 + 2\sum_{\ell=2}^L d_\ell)^2}.$$

For the supremum norm, we infer upon denoting $\mathbf{1}_d = (1, \dots, 1)^\top \in \mathbb{R}^d$ that

$$\sup_{\mathbf{x} \in [0,1]^d} \big|f(\mathbf{x}) - \|\mathbf{x}\|_2^2\big| \geq \sup_{t \in [0,1]} \big|f(t\mathbf{1}_d) - t^2 d\big| = d \sup_{t \in [0,1]} |d^{-1} f(t\mathbf{1}_d) - t^2|.$$

Since $t \mapsto d^{-1} f(t\mathbf{1})$ is a univariate (standard or leaky) ReLU ICNN on $[0,1]$ with L hidden layers and d_ℓ neurons in layer ℓ, the asserted lower bound also follows from Theorem 3.2. □

Proof of Theorem A.2. We write $\mathbf{x} = (x_1, \dots, x_d)^\top$. We define $\kappa : \mathbb{N} \to 2\mathbb{N}_0$ as

$$\kappa(n) := \begin{cases} n & n \text{ is even}, \\ n-1 & n \text{ is odd}. \end{cases}$$

Consider arbitrary $p, q \in \mathbb{N}$ with $pq \geq d$. By Theorem 3.3, there exists a HyCNN $g : \mathbb{R} \to \mathbb{R}$ with $\lfloor L/p \rfloor$ hidden layers, $\kappa(\lfloor (m-1)/q \rfloor)$ neurons per layer and $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$, such that

$$\sup_{x \in [0,1]} |g(x) - x^2| \leq \frac{1}{7.99} \kappa\left(\left\lfloor \frac{m-1}{q} \right\rfloor\right)^{-2\lfloor L/p \rfloor} \leq \frac{1}{7.99} \left\lfloor \frac{m-1}{q} - 1 \right\rfloor^{-2\lfloor L/p \rfloor}. \tag{30}$$

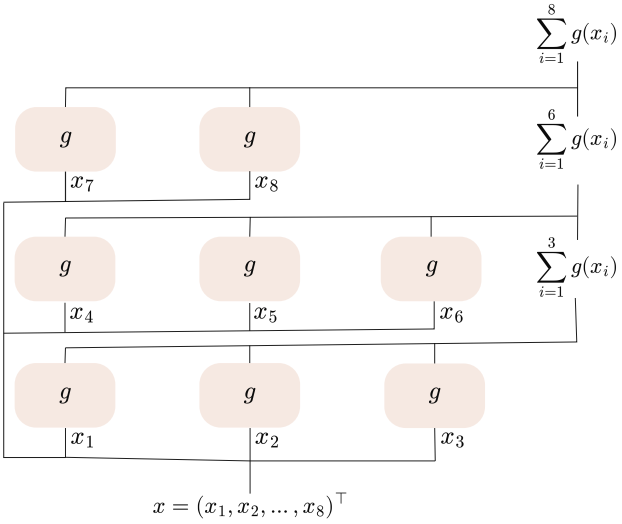

Figure 7: Illustration of the proof of Theorem A.2 for $d = 8$ and $p = q = 3$.

We construct a HyCNN $f\colon \mathbb{R}^d \to \mathbb{R}$ with L hidden layers and m neurons per layer as follows. For each $j \in [p]$, we approximate $x^2_{(j-1)q+1}, \dots, x^2_{jq}$ by applying q parallel copies of the HyCNN g in the hidden layers from $((j-1)\lfloor L/p \rfloor + 1)$ to $j\lfloor L/p \rfloor$ of f. This is possible as the network input is accessible in each hidden layer. If $pq > d$, we apply the construction only up to index d. This construction requires $p\lfloor L/p \rfloor \le L$ layers and $q\kappa(\lfloor (m-1)/q \rfloor) \le m-1$ neurons per layer.

It remains to sum the outputs $g(x_1), \dots, g(x_d)$. This can be done by adding one neuron per hidden layer that stores the sum of the approximated square functions using

$$\overline{\sigma}_\tau\left(\sum_{i=1}^{k} g(x_i) - \tau \log 2, \sum_{i=1}^{k} g(x_i) - \tau \log 2\right) = \sum_{i=1}^{k} g(x_i), \qquad \forall k \in [d].$$

See Figure 7 for an illustration. The HyCNN output $f(\mathbf{x})$ is then $\sum_{i=1}^{d} g(x_i)$. By the triangle inequality and (30), the approximation error is upper bounded by

$$\sup_{\mathbf{x}\in[0,1]^d} \left|f(\mathbf{x}) - \|\mathbf{x}\|_2^2\right| \le \sum_{i=1}^{d} \sup_{x_i\in[0,1]} \left|g(x_i) - x_i^2\right| \le \frac{d}{7.99} \inf_{p,q\in\mathbb{N}:pq\ge d} \left\lfloor \frac{m-1}{q} - 1 \right\rfloor^{-2\lfloor L/p \rfloor}.$$

□

Proof of Theorem A.3. Let $c_k := f_0^{(k)}(0)/k!$, so that $c_k \ge 0$ for all $k \ge 2$. By the Vivanti–Pringsheim theorem, the Taylor series of f_0 at the origin has radius of convergence strictly greater than r. Hence,

$$f_0(x) = \sum_{k=0}^{\infty} c_k x^k, \qquad x \in [0, r].$$

For an integer $q \ge 2$ which will be specified later, define

$$P_q(x) := \sum_{k=0}^{q-1} c_k x^k, \qquad R_q(x) := \sum_{k=q}^{\infty} c_k x^k.$$

It follows

$$\sup_{x\in[0,1]} |R_q(x)| = \sum_{k=q}^{\infty} c_k \le r^{-q} \sum_{k=q}^{\infty} c_k r^k \le r^{-q}\big(f_0(r) - f_0(0) - r f_0'(0)\big).$$

Using Theorem 3.5, we construct $q-2$ HyCNNs $(h_k)_{k=2,\dots,q-1}$, where $h_k\colon \mathbb{R} \to \mathbb{R}$ uses $\overline{\sigma}(a_1, a_2) = \overline{\sigma}_\tau(a_1, a_2)$, has $\lceil \log_2(k) \rceil L$ layers and m neurons per layer and satisfies

$$\sup_{x\in[0,1]} \left|h_k(x) - x^k\right| \le \frac{k}{1.99}(m-1)^{-2L}.$$

After padding these $q-2$ HyCNNs with identity layers (using $\overline{\sigma}_\tau(a-\tau\log 2, a-\tau\log 2)=a$) to equalize their depths, we combine them in parallel. Then, we can construct a HyCNN $h:\mathbb{R}\to\mathbb{R}$ with $\overline{\sigma}(a,b)=\overline{\sigma}_\tau(a,b)$, $\lceil\log_2(q-1)\rceil L$ layers and mq neurons per layer such that $h(x)=c_0+c_1x+\sum_{k=2}^{q-1}c_k h_k(x)$. By choosing $q=2L$, we get the assertion by

$$\begin{aligned}\sup_{x\in[0,1]}\left|h(x)-f_0(x)\right| &\le \sup_{x\in[0,1]}\left|R_q(x)\right|+\sup_{x\in[0,1]}\left|h(x)-P_q(x)\right|\\ &\lesssim r^{-2L}+\sum_{k=2}^{q-1}c_k\sup_{x\in[0,1]}|h_k(x)-x^k|\\ &\lesssim r^{-2L}+(m-1)^{-2L}\sum_{k=2}^{\infty}kc_k\\ &\lesssim r^{-2L}+(m-1)^{-2L},\end{aligned}$$

where we use $\sum_{k=2}^{\infty}kc_k=f_0'(1)-f_0'(0)$ for the last inequality. □

Proof of Proposition A.4. We write $h:=f-g$ and $\varepsilon:=\|h\|_{L^\infty(K')}$. Fix $\mathbf{x}\in K$, a unit vector $\mathbf{u}\in\mathbb{R}^d$, and $r\in(0,1)$, and let $\mathbf{y}:=\mathbf{x}+r\mathbf{u}\in K'$. By convexity of f and M-smoothness of g, we have

$$f(\mathbf{y})\ge f(\mathbf{x})+r\langle\nabla f(\mathbf{x}),\mathbf{u}\rangle,\qquad g(\mathbf{y})\le g(\mathbf{x})+r\langle\nabla g(\mathbf{x}),\mathbf{u}\rangle+\frac{M}{2}r^2.$$

Subtracting the second inequality from the first inequality yields

$$h(\mathbf{y})-h(\mathbf{x})\ge r\langle\nabla h(\mathbf{x}),\mathbf{u}\rangle-\frac{M}{2}r^2.$$

Since $h(\mathbf{y})-h(\mathbf{x})\le 2\varepsilon$, we obtain

$$\langle\nabla h(\mathbf{x}),\mathbf{u}\rangle\le\frac{2\varepsilon}{r}+\frac{Mr}{2}.$$

If $\nabla h(\mathbf{x})\neq 0$, choosing $\mathbf{u}=\nabla h(\mathbf{x})/\|\nabla h(\mathbf{x})\|_2$ gives

$$\|\nabla h(\mathbf{x})\|_2\le\frac{2\varepsilon}{r}+\frac{Mr}{2},\qquad 0<r<1.$$

This bound also holds trivially when $\nabla h(\mathbf{x})=0$.

If $\varepsilon=0$, letting $r\downarrow 0$ gives $\nabla h(\mathbf{x})=\mathbf{0}_d$. If $0<\varepsilon\le M/4$, choosing $r=2\sqrt{\varepsilon/M}\le 1$ gives

$$\|\nabla h(\mathbf{x})\|_2\le 2\sqrt{M\varepsilon}.$$

Otherwise, if $\varepsilon>M/4$, letting $r\uparrow 1$ gives

$$\|\nabla h(\mathbf{x})\|_2\le 2\varepsilon+\frac{M}{2}\le 4\varepsilon.$$

Taking the supremum over $\mathbf{x}\in K$, we conclude that

$$\|\nabla f-\nabla g\|_{L^\infty(K)}=\|\nabla h\|_{L^\infty(K)}\le 2\sqrt{M\varepsilon}+4\varepsilon.$$

Finally, since K has unit Lebesgue measure,

$$\|\nabla f-\nabla g\|_{L^p(K)}\le\|\nabla f-\nabla g\|_{L^\infty(K)}$$

for every $p\in[1,\infty)$. □

C Derivation of the Initialization Scheme for HyCNNs

In this section, we derive the initialization scheme introduced in Section 2, closely following the steps for the ICNN initialization proposed by Hoedt & Klambauer (2023) but appropriately adapted for HyCNNs.

For $\ell \in [L-1]$, we rewrite the HyCNN layer (2) as

$$\begin{aligned}\mathbf{s}_{\ell+1} &:= V_\ell^1 \mathbf{z}_\ell + W_\ell^1 \mathbf{x} + \mathbf{b}_\ell^1,\\ \mathbf{t}_{\ell+1} &:= V_\ell^2 \mathbf{z}_\ell + W_\ell^2 \mathbf{x} + \mathbf{b}_\ell^2,\\ \mathbf{z}_{\ell+1} &:= \max(\mathbf{s}_{\ell+1}, \mathbf{t}_{\ell+1}).\end{aligned}$$

We write $\mathbf{x} = (x_i)_{i\in[d]}$ and for $k \in \{1,2\}$,

$$\mathbf{s}_\ell = (s_{\ell,i})_{i\in[d_\ell]}, \quad \mathbf{t}_\ell = (t_{\ell,i})_{i\in[d_\ell]}, \quad \mathbf{z}_\ell = (z_{\ell,i})_{i\in[d_\ell]}, \quad \text{for all } \ell \in [L],$$
$$V_\ell^k = (v_{\ell,i,j}^{(k)})_{i\in[d_{\ell+1}],j\in[d_\ell]}, \quad W_\ell^k = (w_{\ell,i,j}^{(k)})_{i\in[d_{\ell+1}],j\in[d]}, \quad \mathbf{b}_\ell^k = (b_{\ell,i}^{(k)})_{i\in[d_{\ell+1}]} \quad \text{for all } \ell \in [L-1].$$

At initialization, the weight parameters are drawn independently with distributions ($\ell \in [L-1]$),

$$\begin{aligned} v_{\ell,i,j}^{(k)} &\sim \mathcal{D}_\ell, && k \in \{1,2\},\ i \in [d_{\ell+1}],\ j \in [d_\ell],\\ w_{\ell,i,j}^{(k)} &\sim N(0, \sigma_{w,\ell}^2), && k \in \{1,2\},\ i \in [d_{\ell+1}],\ j \in [d],\\ b_{\ell,i}^{(k)} &\sim N(\mu_{b,\ell}, \sigma_{b,\ell}^2), && k \in \{1,2\},\ i \in [d_{\ell+1}],\end{aligned}$$

where $\mathcal{D}_\ell$ is a distribution in a pre-specified class with support on $\mathbb{R}_+$. Denote by $\mu_{v,\ell}$ and $\sigma_{v,\ell}^2$ the expectation and the variance of $\mathcal{D}_\ell$, respectively. For each $\ell \in [L]$, the $2d_\ell$ random variables $s_{\ell,1}, t_{\ell,1}, \ldots, s_{\ell,d_\ell}, t_{\ell,d_\ell}$ are exchangeable. It follows that $\mathbb{E}(s_{\ell,1}) = \mathbb{E}(s_{\ell,i}) = \mathbb{E}(t_{\ell,i})$, $\mathbb{E}(s_{\ell,1}^2) = \mathbb{E}(s_{\ell,i}^2) = \mathbb{E}(t_{\ell,i}^2)$ and $\mathbb{E}(s_{\ell,1}t_{\ell,1}) = \mathbb{E}(s_{\ell,i}s_{\ell,i'}) = \mathbb{E}(t_{\ell,i}t_{\ell,i'}) = \mathbb{E}(s_{\ell,i}t_{\ell,i}) = \mathbb{E}(s_{\ell,i}t_{\ell,i'})$ for any $i, i' \in [d_\ell]$ with $i \neq i'$. Moreover, $\mathbb{E}(z_{\ell,1}) = \mathbb{E}(z_{\ell,i})$, $\mathbb{E}(z_{\ell,1}^2) = \mathbb{E}(z_{\ell,i}^2)$, and $\mathbb{E}(z_{\ell,1}z_{\ell,2}) = \mathbb{E}(z_{\ell,i}z_{\ell,i'})$.

We now fix a layer $\ell \in [L-1]$. By definition, for $i \in [d_{\ell+1}]$ we have

$$s_{\ell+1,i} = \sum_{j=1}^{d_\ell} v_{\ell,i,j}^{(1)} z_{\ell,j} + \sum_{j=1}^{d} w_{\ell,i,j}^{(1)} x_j + b_{\ell,i}^{(1)}, \tag{31}$$

$$t_{\ell+1,i} = \sum_{j=1}^{d_\ell} v_{\ell,i,j}^{(2)} z_{\ell,j} + \sum_{j=1}^{d} w_{\ell,i,j}^{(2)} x_j + b_{\ell,i}^{(2)} \tag{32}$$

with

$$z_{\ell,j} = s_{\ell,j} \vee t_{\ell,j}.$$

The key objective is to ensure that pre-activations exhibit comparable statistical properties across all layers at initialization. To this end, we derive fixed-point equations for the mean, variance, and covariance,

$$\mathbb{E}(s_{\ell+1,1}) = \mathbb{E}(s_{\ell,1}), \quad \mathbb{E}(s_{\ell+1,1}^2) = \mathbb{E}(s_{\ell,1}^2), \quad \mathbb{E}(s_{\ell+1,1}t_{\ell+1,1}) = \mathbb{E}(s_{\ell,1}t_{\ell,1}), \tag{33}$$

and identify suitable hyperparameter configurations $(\mu_{v,\ell}, \sigma_{v,\ell}^2, \sigma_{w,\ell}^2, \mu_{b,\ell}, \sigma_{b,\ell}^2)$. For simplicity, we drop the subscript ℓ from the hyperparameters below. From (31) and (32),

$$\begin{aligned}&\mathbb{E}(s_{\ell+1,1}) = d_\ell \mu_v \mathbb{E}(z_{\ell,1}) + \mu_b,\\ &\mathbb{E}(s_{\ell+1,1}^2) = \sigma_b^2 + \mu_b^2 + \sigma_w^2 \|\mathbf{x}\|_2^2 + d_\ell(\mu_v^2 + \sigma_v^2)\mathbb{E}(z_{\ell,1}^2) + d_\ell(d_\ell - 1)\mu_v^2 \mathbb{E}(z_{\ell,1}z_{\ell,2}) + 2d_\ell \mu_b \mu_v \mathbb{E}(z_{\ell,1}),\\ &\mathbb{E}(s_{\ell+1,1}t_{\ell+1,1}) = \mu_b^2 + d_\ell \mu_v^2 \mathbb{E}(z_{\ell,1}^2) + d_\ell(d_\ell - 1)\mu_v^2 \mathbb{E}(z_{\ell,1}z_{\ell,2}) + 2d_\ell \mu_b \mu_v \mathbb{E}(z_{\ell,1}).\end{aligned}$$

Assume we already picked hyperparameters in the previous layer so that $\mathbb{E}(s_{\ell,1}) = 0$. Choosing $\mu_b = -d_\ell \mu_v \mathbb{E}(z_{\ell,1})$ ensures then $\mathbb{E}(s_{\ell+1,1}) = 0$, $\mu_b^2 + 2d_\ell \mu_v \mu_b \mathbb{E}(z_{\ell,1}) = -d_\ell^2 \mu_v^2 \mathbb{E}(z_{\ell,1})^2$, and

$$\mathbb{E}(s_{\ell+1,1}^2) = \sigma_b^2 + \sigma_w^2 \|\mathbf{x}\|_2^2 + d_\ell(\mu_v^2 + \sigma_v^2)\mathbb{E}(z_{\ell,1}^2) + d_\ell(d_\ell - 1)\mu_v^2 \mathbb{E}(z_{\ell,1}z_{\ell,2}) - d_\ell^2 \mu_v^2 \mathbb{E}(z_{\ell,1})^2, \tag{34}$$

$$\mathbb{E}(s_{\ell+1,1}t_{\ell+1,1}) = d_\ell \mu_v^2 \mathbb{E}(z_{\ell,1}^2) + d_\ell(d_\ell - 1)\mu_v^2 \mathbb{E}(z_{\ell,1}z_{\ell,2}) - d_\ell^2 \mu_v^2 \mathbb{E}(z_{\ell,1})^2. \tag{35}$$

We approximate the joint distribution of the random variables $s_{\ell,1}, \ldots, s_{\ell,d_\ell}, t_{\ell,1}, \ldots, t_{\ell,d_\ell}$ with a Gaussian distribution with the same mean and covariance matrix. Define $A := \mathbb{E}(s_{\ell,1}^2)$ and

$B := \mathbb{E}(s_{\ell,1}t_{\ell,1})$. Applying Lemma C.1 with $(X,Y,Z,W) = (s_{\ell,1}, t_{\ell,1}, s_{\ell,2}, t_{\ell,2})$, $\sigma^2 = A$, and $\rho = B/A$, we have

$$\mathbb{E}(z_{\ell,1}) = \mathbb{E}(s_{\ell,1} \vee t_{\ell,1}) = \sqrt{\frac{A-B}{\pi}}, \tag{36}$$

$$\mathbb{E}(z_{\ell,1}^2) = \mathbb{E}\big((s_{\ell,1} \vee t_{\ell,1})^2\big) = A, \tag{37}$$

$$\mathbb{E}(z_{\ell,1}z_{\ell,2}) = \mathbb{E}\big((s_{\ell,1} \vee t_{\ell,1})(s_{\ell,2} \vee t_{\ell,2})\big) = B + \frac{A-B}{\pi}. \tag{38}$$

Now, we solve the equations (33). By plugging (36), (37) and (38) into (34), we obtain

$$\begin{aligned} A = \mathbb{E}(s_{\ell+1,1}^2) &= \sigma_b^2 + \sigma_w^2\|\mathbf{x}\|_2^2 + d_\ell(\mu_v^2+\sigma_v^2)A + d_\ell(d_\ell-1)\mu_v^2\left(B + \frac{A-B}{\pi}\right) - d_\ell^2\mu_v^2\frac{A-B}{\pi} \\ &= \sigma_b^2 + \sigma_w^2\|\mathbf{x}\|_2^2 + d_\ell(\mu_v^2+\sigma_v^2)A + d_\ell(d_\ell-1)\mu_v^2 B - d_\ell\mu_v^2\frac{A-B}{\pi}. \end{aligned} \tag{39}$$

Similarly, by plugging (36), (37) and (38) into (35), we obtain

$$B = \mathbb{E}(s_{\ell+1,1}t_{\ell+1,1}) = d_\ell\mu_v^2 A + d_\ell(d_\ell-1)\mu_v^2 B - d_\ell\mu_v^2\frac{A-B}{\pi}. \tag{40}$$

Using $\rho = B/A$, (40) yields

$$\mu_v^2 = \frac{B}{d_\ell A + d_\ell(d_\ell-1)B - \frac{d_\ell(A-B)}{\pi}} = \frac{\rho}{d_\ell + d_\ell(d_\ell-1)\rho - \frac{d_\ell(1-\rho)}{\pi}}.$$

Also, by subtracting (40) from (39) we get

$$A(1-\rho) = \sigma_b^2 + \sigma_w^2\|\mathbf{x}\|_2^2 + d_\ell\sigma_v^2 A.$$

Since the extreme cases $\rho = 0 = \mu_v^2$ and $\rho = 1$ are undesirable, we choose $\rho = \frac{1}{2}$. It follows

$$\mu_v = \sqrt{\frac{1}{d_\ell^2 + (1-\frac{1}{\pi})d_\ell}}$$

and

$$\frac{1}{2}A = \sigma_b^2 + \sigma_w^2\|\mathbf{x}\|_2^2 + d_\ell\sigma_v^2 A.$$

Provided that $\|\mathbf{x}\|_2^2 \approx d$, a reasonable choice for other hyperparameters would be $\sigma_b^2 = 0$, $\sigma_w^2 = 1/(4d)$ and $\sigma_v^2 = 1/(4d_\ell)$. Then we get $A = 1$ and $B = \frac{1}{2}$. Finally, we obtain

$$\mu_b = -d_\ell\mu_v\mathbb{E}(z_{\ell,1}) = -d_\ell\sqrt{\frac{1}{d_\ell^2 + (1-\frac{1}{\pi})d_\ell}}\sqrt{\frac{1}{2\pi}} = -\sqrt{\frac{d_\ell}{2\pi d_\ell + 2\pi - 2}}.$$

Lemma C.1. *If $\rho \in (-1,1)$ and*

$$\begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \sigma^2\begin{pmatrix} 1 & \rho & \rho & \rho \\ \rho & 1 & \rho & \rho \\ \rho & \rho & 1 & \rho \\ \rho & \rho & \rho & 1 \end{pmatrix}\right),$$

then,

$$\mathbb{E}[X \vee Y] = \sigma\sqrt{\frac{1-\rho}{\pi}}, \quad \mathbb{E}[(X \vee Y)^2] = \sigma^2, \quad \mathbb{E}[(X \vee Y)(Z \vee W)] = \left(\rho + \frac{1-\rho}{\pi}\right)\sigma^2.$$

Proof. We define

$$S_1 := X + Y, \quad S_2 := Z + W, \quad D_1 := X - Y, \quad D_2 := Z - W.$$

Since $(S_1, S_2, D_1, D_2)^\intercal$ is jointly Gaussian and $\mathrm{Var}(X) = \mathrm{Var}(Y) = \mathrm{Var}(Z) = \mathrm{Var}(W) = \sigma^2$, we have $S_1 \perp D_1, S_1 \perp D_2, S_2 \perp D_1, S_2 \perp D_2, D_1 \perp D_2$ and

$$X \vee Y = \frac{S_1 + |D_1|}{2}, \quad Z \vee W = \frac{S_2 + |D_2|}{2}.$$

The absolute value of a standard Gaussian random variable has mean $\sqrt{2/\pi}$. From $S_1, S_2 \sim N(0, 2\sigma^2(1+\rho))$ and $D_1, D_2 \sim N(0, 2\sigma^2(1-\rho))$, we get

$$\mathbb{E}[X \vee Y] = \mathbb{E}\left[\frac{S_1 + |D_1|}{2}\right] = \frac{1}{2}\sqrt{\frac{4\sigma^2(1-\rho)}{\pi}} = \sigma\sqrt{\frac{1-\rho}{\pi}}$$

and

$$\mathbb{E}\big((X \vee Y)^2\big) = \mathbb{E}\left(\frac{S_1^2 + D_1^2 + 2S_1|D_1|}{4}\right) = \frac{2\sigma^2(1+\rho) + 2\sigma^2(1-\rho)}{4} = \sigma^2.$$

We derive the last statement from

$$\begin{aligned}\mathbb{E}\big((X \vee Y)(Z \vee W)\big) &= \mathbb{E}\left(\frac{S_1S_2 + S_1|D_2| + S_2|D_1| + |D_1||D_2|}{4}\right) \\ &= \mathbb{E}\left(\frac{(X+Y)(Z+W)}{4}\right) + \frac{1}{4}\mathbb{E}(|D_1|)(|D_2|) \\ &= \rho\sigma^2 + \frac{1-\rho}{\pi}\sigma^2.\end{aligned}$$

□

D Additional Experiments

In this section, we provide additional experimental results and implementation details complementing the experiments described in Section 5. The section is organized as follows. In Section D.1, we visualize randomly initialized HyCNNs. In Section D.2, we present additional regression experiments, including algorithmic details, univariate regression fits, results across different sample sizes and dimensions, and detailed performance tables. In Section D.3, we present the HyCNN-based OT map estimation procedure, simulation settings, and detailed performance tables. All experiments were run on a MacBook Pro 16" with Apple M4 Pro with 24 GB of unified memory and 512 GB of storage using the integrated GPU. ChatGPT Codex 5.4 and later 6.1 Sol as well as Claude Code Opus 4.6 were used to assist in implementation and debugging, and generated code was carefully reviewed and tested by the authors. The code to some experiments is provided on GitHub at https://www.github.com/hundrieser/HyCNN.

Beyond the experiments reported in the paper, the full research project required additional compute for preliminary and exploratory work that is not included in the final manuscript. This includes the development and validation of the HyCNN initialization scheme, hyperparameter exploration (e.g., learning rate, batch size, number of epochs, width, and depth), ablations on skip-connection variants, as well as failed runs and debugging. All such runs were executed on the same hardware described above. We estimate that the total compute consumed during the project exceeded that required to reproduce the reported experiments by approximately a factor of three to five.

D.1 Visualization of HyCNN initialization

Following our initialization scheme from Section 2, we display the surfaces of randomly initialized HyCNNs across different depths in Figure 8 for input dimension $d = 2$ (i.e. $\mathbf{x} \in [-10, 10]^2$) and width $m = 32$. Upon close inspection, we see that shallow networks behave more polyhedral, while deeper networks exhibit more smoothness. Moreover, the surfaces also do not explode at initialization as the depth increases, and are roughly centered around the origin.

D.2 Regression experiments

In this section, we provide details and additional simulations to the regression experiments described in Section 5. We first describe the experimental setup (Section D.2.1), then present univariate regression fits (Section D.2.2), 50-dimensional regression across different sample sizes (Section D.2.3), regression across different dimensions (Section D.2.4), and interpolation across different dimensions (Section D.2.5). Appendices D.2.6 and D.2.7 present ablation studies investigating the effects of the smoothing parameter on propagation and regression, and assess the effect of a multi-lane architecture compared to the two-lane HyCNN framework.

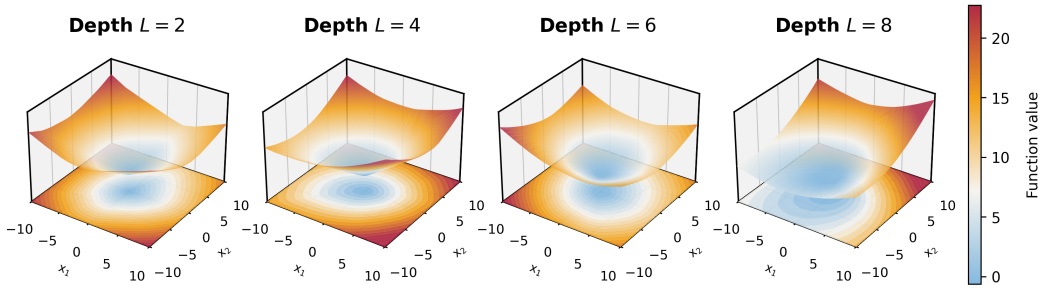

Figure 8: HyCNNs at initialization across different depths. Each panel shows a single randomly initialized HyCNN according to the initialization scheme from Section 2 with maxout activation, input dimension $d = 2$, domain $\mathbf{x} \in [-10, 10]^2$, width $m = 32$, and depths $L \in \{2, 4, 6, 8\}$. The surface plot is shown together with its contour projection on the base plane. The visualization demonstrates that the initialized functions remain bounded and roughly centered around the origin as depth increases, and that deeper networks produce smoother surfaces compared to the more polyhedral shallow networks.

D.2.1 Experimental setup

In total, we consider the following ground-truth functions:

(a) $f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2$, (b) $f_2(\mathbf{x}) = \|\mathbf{x}\|_4^4$, (c) $f_3(\mathbf{x}) = \|\mathbf{x}\|_2^2 + 0.25 \sin(20\|\mathbf{x}\|_2)$,

(d) $f_4(\mathbf{x}) = \|\mathbf{x}\|_1$, (e) $f_5(\mathbf{x}) = \max\left(\|\mathbf{x} - \boldsymbol{\mu}\|_2^2, \|\mathbf{x} + \boldsymbol{\mu}\|_2^2\right)$, (f) $f_6(\mathbf{x}) = \exp(\|\mathbf{x}\|_1/\sqrt{d})$,

where for every generated sample, we first draw the entries of the vector $\boldsymbol{\mu} \in \mathbb{R}^d$ independently from $N(0, 1/d)$.

Data generation. We draw training samples $(\mathbf{x}_i, f(\mathbf{x}_i) + \epsilon_i)_{i=1}^n$ with $\mathbf{x}_i$ drawn independently from $\mathrm{Unif}([-1, 1]^d)$ and the noise ϵ_i drawn independently from $N(0, \sigma^2)$ for $\sigma^2 \geq 0$.

Network architectures. We compare MLPs with ReLU activation, ICNNs with ReLU activation, GroupMax networks (GMNs, Warin 2024, which correspond to maxout HyCNNs without skip connections) and HyCNNs. For each architecture, we consider networks of depth $L = 2, 4, 8, 16, 32$. Across the following settings, for small and large dimensions, we adjust the network widths to ensure a comparable number of trainable parameters across architectures.

Training. For each architecture $\widehat{f}_\theta$ we first normalize the data per coordinate by subtracting the empirical mean and dividing by the empirical standard deviation along each input dimension. Each network is then trained on the standardized data, and finally the trained models are destandardized to compute the reported MSE in the original units. For training, we perform empirical risk minimization with respect to the mean-squared-error loss, i.e., we minimize

$$\theta \mapsto \frac{1}{n} \sum_{i=1}^{n} \left(\widehat{f}_\theta(\mathbf{x}_i) - y_i\right)^2,$$

over the full training set $(\mathbf{x}_i, y_i)_{i=1}^n$, where $y_i = f(\mathbf{x}_i) + \epsilon_i$. Optimization is carried out with the Adam optimizer (Kingma & Ba, 2015) with learning rate $\lambda = 10^{-2}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, batch size $\min(n, 1{,}000)$, and 100 epochs, applied jointly to all trainable parameters. For the ICNN and HyCNN architectures, the hidden-to-hidden weight matrices V_ℓ must remain elementwise non-negative throughout training, as this is required for the resulting network to be input-convex in $\mathbf{x}$. We impose this constraint via a smooth reparametrization of the weights: we store an unconstrained parameter tensor $R_\ell \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$ and use, on every forward pass, the effective weight $V_\ell = \mathrm{softplus}(R_\ell) = \log(1 + \exp(R_\ell))$, applied entrywise. Because $\mathrm{softplus}: \mathbb{R} \to (0, \infty)$ is smooth and strictly positive, the entries of V_ℓ are automatically non-negative for every value of R_ℓ, and gradients flow back through the softplus so that Adam can be applied to the unconstrained R_ℓ directly. All other parameters (input-to-hidden weights, biases, quadratic-skip matrices where applicable) are unconstrained real-valued parameters and are updated without any reparametrization.

For initialization, we use for the HyCNNs the scheme described in Section 2 (the initialization acts directly on V_ℓ and not on R_ℓ), for the ICNN we use the initialization scheme of Hoedt & Klambauer (2023), for GMNs we use the same initialization scheme as for HyCNNs while setting the skip connection weights to zero, and for MLPs we use PyTorch's default initialization scheme for `nn.Linear` layers, which draws each weight entry independently from $\mathrm{Uniform}(-1/\sqrt{d_{\mathrm{in}}}, 1/\sqrt{d_{\mathrm{in}}})$, where d_{in} denotes the number of input features to the respective layer; the bias vector is sampled from the same distribution. Internally, PyTorch implements this via `kaiming_uniform_(a=math.sqrt(5))` (He et al., 2015), which with $a = \sqrt{5}$ reduces the Kaiming bound $\sqrt{6/((1+a^2)\cdot d_{\mathrm{in}})}$ to $1/\sqrt{d_{\mathrm{in}}}$.

Evaluation. Given a trained network $\widehat{f}$, we evaluate the prediction MSE on an independent test set $(\widetilde{\mathbf{x}}_j)_{j=1}^{N_{\mathrm{test}}}$ of size $N_{\mathrm{test}} = 1{,}000$ drawn from $\mathrm{Unif}([-1,1]^d)$ (i.e., the same distribution as the training covariates but without noise), defined as

$$\mathrm{MSE}_{\mathrm{pred}} := \frac{1}{N_{\mathrm{test}}} \sum_{j=1}^{N_{\mathrm{test}}} \left(f(\widetilde{\mathbf{x}}_j) - \widehat{f}(\widetilde{\mathbf{x}}_j) \right)^2, \tag{41}$$

where f denotes the ground-truth function. Each simulation run, except for the univariate regression which is only conducted once for visualization, is repeated 10 times with different random seeds for the training sample generation and the network initialization.

D.2.2 Univariate regression

For univariate regression ($d = 1$), we visualize the fits of the different architectures across different depths for the ground-truth functions (a–f) and different noise levels. Figure 9 shows the best MLP, ICNN and GMN across the different depths, and the outcome of different HyCNN architectures. The widths of the networks are chosen as 64 for the MLP, 64 for the ICNN, 46 for the GMN, and 44 for the HyCNNs to ensure similar parameter counts across the different architectures.

We see that deeper HyCNNs often exhibit a better fit than more shallow HyCNNs. Except for the function f_3 which is non-convex, the best MLP, ICNN and GMN across the different depths often exhibit a worse fit than the best HyCNN across the different depths. Moreover, for the non-convex function f_3, the best convex shape-constrained network is given by a HyCNN.

D.2.3 Regression in dimension $d = 50$ across different sample sizes

For the same network classes, i.e., MLPs, ICNNs, GMNs, and HyCNNs, with candidate depths $L \in \{2, 4, 8, 16\}$, we now consider for dimension 50 the same functions (a–f) and solve regression problems from the Gaussian noise model across different sample sizes $n \in \{100, 200, 500, 1{,}000, 2{,}000, 5{,}000, 10{,}000\}$. The widths of the MLPs are chosen as 84, the ICNNs as 64, the GMNs as 60, and the HyCNNs as 40, to ensure similar parameter counts across the different architectures, see Table 1. The resulting prediction MSEs are visualized in Figures 4 and 10.

We see that deeper HyCNNs exhibit a better fit than more shallow HyCNNs across larger sample sizes $n \geq 500$. Moreover, the best MLP, ICNN and GMN across the different depths often exhibit a worse fit than most HyCNNs across depths $L \geq 4$, especially for larger sample sizes. In particular, even for the non-convex function f_3, the best network is nonetheless a HyCNN, which is likely due to the fact that f_3 is well approximated by a convex function. For sample sizes several orders of magnitude larger, it is likely that the MLP will perform best, however for the sample sizes considered here, it seems that the HyCNNs are able to more effectively leverage the inductive bias of convexity to achieve better generalization performance than the MLPs.

D.2.4 Regression across different dimensions for sample size $n = 5{,}000$

We also computed the prediction MSE for the same network classes with depths $L \in \{2, 4, 8, 16, 32\}$ and widths as in Appendix D.2.3 for the regression tasks under the same functions (a–f) across different dimensions $d \in \{1, 2, 5, 10, 20, 50\}$ for a fixed sample size of $n = 5{,}000$ in the noisy setting with additive Gaussian noise ($\sigma = 1$ for f_1, f_2, f_4, f_5, f_6 and $\sigma = 0$ for f_3). The results are reported in Table 2 (f_1), Table 3 (f_2), Table 4 (f_3), Table 5 (f_4), Table 6 (f_5), and Table 7 (f_6).

Across all settings, there are some consistent observations. HyCNNs with depth at least four typically achieve the lowest prediction MSE, particularly in higher dimensions ($d \geq 10$), and in most examples

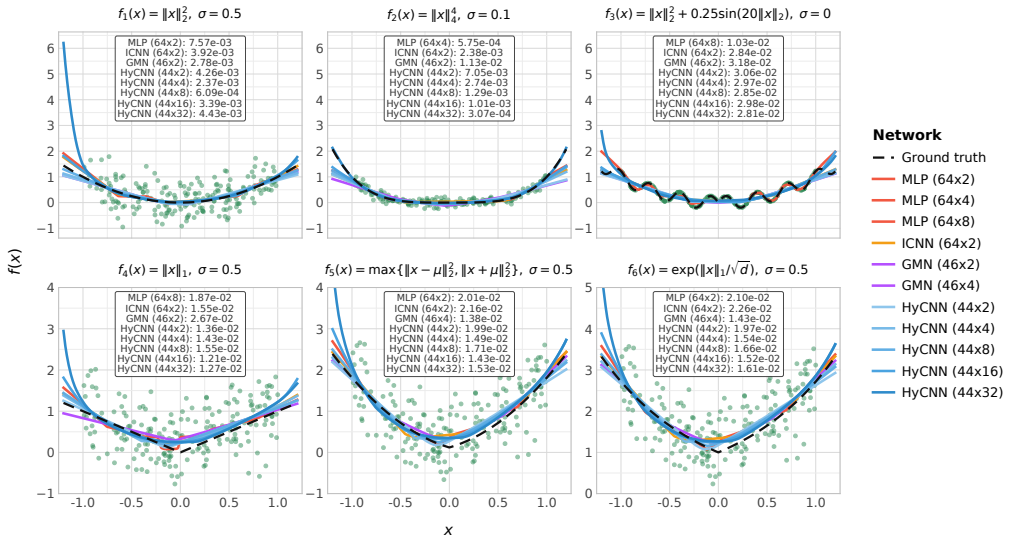

Figure 9: Univariate regression results ($d = 1$) for the six ground-truth functions (a–f). Each panel shows HyCNNs of width 44 and depths $L \in \{2, 4, 8, 16, 32\}$, together with the best MLP (width 64), ICNN (width 64), and GMN (width 46) among the same depths. Green dots show training samples and black dashed curves show the ground-truth functions. Each panel uses one run with $n = 200$ samples and the original sample and method seeds. The noise standard deviations are $\sigma = 0.5$ for f_1, f_4, f_5, f_6, $\sigma = 0.1$ for f_2, and $\sigma = 0$ for f_3. The insets report the prediction MSE on 1,000 independent noiseless targets.

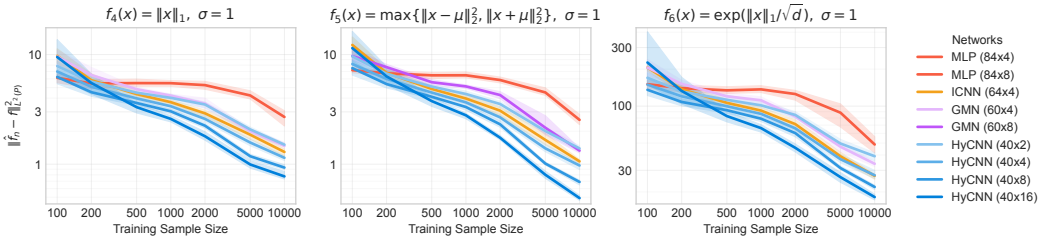

Figure 10: Prediction MSE for different training sample sizes n for the 50-dimensional regression tasks with 10th-90th percentile bands across 10 runs. Each panel shows HyCNNs with depths $2, 4, 8, 16$ and the best MLP, ICNN, GMN at $n = 10^4$ among similar depths.

they also consistently outperform both ICNNs and MLPs. GMNs are occasionally competitive with HyCNNs at moderate depths, which may be attributed to their implicit regularization. Deeper HyCNNs tend to improve performance as the dimension increases, with depth $L = 16$ often providing the best results for $d = 50$. Very deep architectures ($L = 32$) can exhibit instabilities due to occasional training failures, as reflected in the larger standard errors. In contrast, ICNNs and GMNs degrade substantially beyond depth 8, specifically ICNNs at depth 32 often exhibit numerical instabilities (MSE $> 10^{11}$), and GMNs at depth 32 consistently diverge. Since GMNs are HyCNNs without skip connections from the input, this suggests that the skip connections play a critical role in stabilizing training of deeper architectures. MLPs, while trainable at all depths, underperform the convex architectures for most functions and dimensions, reflecting their lack of inductive bias towards convexity. For the non-convex function f_3, the picture is more nuanced: in low dimensions ($d \leq 5$), the MLP performs considerably better than the convex architectures, as it is not constrained to produce convex fits. However, for $d \geq 10$ the HyCNN achieves the best prediction MSE, suggesting that the convexity bias of HyCNNs acts as a beneficial regularizer in higher dimensions, where f_3 is well approximated by a convex function.

Table 1: Number of trainable parameters by architecture, width m, depth L, and input dimension d. Network size is denoted by $(m \times L)$, where L is the number of hidden layers.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	7,393	7,477	7,729	8,149	8,989	11,509
	(84×4)	21,673	21,757	22,009	22,429	23,269	25,789
	(84×8)	50,233	50,317	50,569	50,989	51,829	54,349
	(84×16)	107,353	107,437	107,689	108,109	108,949	111,469
	(84×32)	221,593	221,677	221,929	222,349	223,189	225,709
ICNN	(64×2)	4,418	4,547	4,934	5,579	6,869	10,739
	(64×4)	12,866	13,123	13,894	15,179	17,749	25,459
	(64×8)	29,762	30,275	31,814	34,379	39,509	54,899
	(64×16)	63,554	64,579	67,654	72,779	83,029	113,779
	(64×32)	131,138	133,187	139,334	149,579	170,069	231,539
GMN	(60×2)	7,500	7,620	7,980	8,580	9,780	13,380
	(60×4)	21,900	22,020	22,380	22,980	24,180	27,780
	(60×8)	50,700	50,820	51,180	51,780	52,980	56,580
	(60×16)	108,300	108,420	108,780	109,380	110,580	114,180
	(60×32)	223,500	223,620	223,980	224,580	225,780	229,380
HyCNN	(40×2)	3,562	3,723	4,206	5,011	6,621	11,451
	(40×4)	10,282	10,603	11,566	13,171	16,381	26,011
	(40×8)	23,722	24,363	26,286	29,491	35,901	55,131
	(40×16)	50,602	51,883	55,726	62,131	74,941	113,371
	(40×32)	104,362	106,923	114,606	127,411	153,021	229,851

Table 2: Prediction MSE and standard error (SE) for learning $f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2$ with $n = 5{,}000$ training samples, $\mathbf{X}_i \sim \mathrm{Unif}([-1, 1]^d)$, and Gaussian noise of standard deviation $\sigma = 1$. **All entries for MSE and SE are multiplied by 10^2 for depiction purposes**. Means and SEs are computed across ten independent training runs for independent data; networks are trained for 100 epochs. Widths are 84 for MLP, 64 for ICNN, 60 for GMN, and 40 for HyCNN, with network depths $L \in \{2, 4, 8, 16, 32\}$ to ensure similar parameter counts across different depths at dimension $d = 50$. The smallest mean within each network family and dimension is bold and colored; the three smallest mean prediction errors across all networks are underlined.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	**0.23** (0.03)	**0.97** (0.10)	**10.24** (0.53)	**36.54** (1.91)	**83.12** (2.58)	463.32 (11.16)
	(84×4)	0.35 (0.06)	1.79 (0.15)	38.19 (1.90)	78.06 (2.85)	107.56 (2.53)	432.46 (11.96)
	(84×8)	0.56 (0.10)	1.67 (0.14)	26.07 (2.32)	67.91 (2.20)	117.32 (6.76)	**419.92** (4.74)
	(84×16)	2.32 (1.05)	9.70 (2.64)	26.38 (4.65)	70.49 (6.26)	137.62 (15.72)	424.10 (9.37)
	(84×32)	8.90 (0.07)	17.73 (0.23)	44.41 (0.41)	88.12 (1.18)	174.48 (1.96)	440.19 (4.04)
ICNN	(64×2)	**0.16** (0.04)	**0.41** (0.03)	**2.31** (0.07)	**8.53** (0.31)	32.51 (0.90)	221.73 (4.82)
	(64×4)	1.90 (1.09)	3.81 (2.14)	3.43 (0.92)	16.61 (7.96)	**29.71** (1.10)	**189.36** (27.34)
	(64×8)	8.90 (0.06)	16.18 (1.64)	40.19 (4.00)	88.49 (1.28)	134.74 (18.55)	400.70 (30.23)
	(64×16)	8.97 (0.07)	17.78 (0.24)	44.52 (0.39)	88.43 (1.26)	175.74 (2.09)	445.51 (3.82)
	(64×32)	8.90 (0.06)	17.75 (0.24)	44.52 (0.40)	88.45 (1.27)	175.75 (2.10)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	0.16 (0.03)	0.48 (0.04)	2.22 (0.07)	7.25 (0.24)	26.88 (0.41)	206.77 (4.18)
	(60×4)	0.12 (0.04)	0.37 (0.02)	**2.14** (0.04)	**7.02** (0.29)	**25.75** (0.50)	**173.75** (1.60)
	(60×8)	**0.10** (0.03)	**0.36** (0.02)	2.25 (0.25)	11.45 (1.75)	30.73 (3.14)	213.01 (12.95)
	(60×16)	3.17 (1.45)	5.17 (1.04)	14.41 (0.57)	36.74 (0.99)	91.64 (1.67)	329.94 (2.64)
	(60×32)	1,792 (473)	7,176 (2,797)	3,169 (949)	2,172 (741)	4,615 (2,000)	3,836 (995)
HyCNN	(40×2)	0.14 (0.02)	0.51 (0.04)	2.50 (0.09)	7.91 (0.19)	26.17 (0.48)	184.94 (2.09)
	(40×4)	0.13 (0.02)	0.44 (0.05)	2.05 (0.05)	**6.69** (0.24)	23.90 (0.64)	133.67 (3.29)
	(40×8)	0.12 (0.03)	0.37 (0.03)	2.13 (0.05)	6.84 (0.29)	**20.64** (0.67)	91.16 (1.53)
	(40×16)	**0.10** (0.03)	0.33 (0.03)	**1.91** (0.09)	6.74 (0.45)	21.69 (0.43)	**71.65** (1.75)
	(40×32)	0.10 (0.03)	**0.27** (0.02)	2.01 (0.25)	9.92 (3.10)	75.66 (41.94)	1,017 (855)

Table 3: Prediction MSE (standard error) for different models and dimensions (**all entries multiplied by** 10^2), as in Table 2, but with true function $f_2(\mathbf{x}) = \|\mathbf{x}\|_4^4$.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	**0.26** (0.05)	**0.91** (0.08)	**13.16** (0.67)	**45.95** (1.59)	**107.10** (2.21)	445.50 (11.23)
	(84×4)	0.32 (0.05)	1.80 (0.11)	38.28 (2.12)	91.17 (2.71)	131.48 (2.21)	399.68 (9.83)
	(84×8)	0.61 (0.10)	1.92 (0.26)	31.60 (2.64)	76.21 (4.00)	128.36 (5.18)	402.67 (8.70)
	(84×16)	4.91 (0.92)	6.88 (2.08)	22.88 (3.83)	57.85 (6.62)	140.48 (1.97)	357.13 (5.30)
	(84×32)	7.16 (0.08)	14.37 (0.25)	35.80 (0.41)	70.66 (0.92)	140.48 (1.96)	**351.70** (3.31)
ICNN	(64×2)	**0.19** (0.04)	**0.55** (0.04)	**4.27** (0.08)	**13.52** (0.38)	**39.89** (1.10)	225.95 (3.72)
	(64×4)	1.72 (0.84)	3.28 (1.72)	7.69 (3.12)	19.54 (5.91)	40.27 (1.52)	**190.77** (17.96)
	(64×8)	7.17 (0.08)	13.21 (1.29)	32.72 (3.02)	71.09 (1.00)	114.03 (12.66)	328.51 (18.12)
	(64×16)	7.23 (0.09)	14.41 (0.26)	35.94 (0.40)	71.04 (0.98)	141.49 (2.04)	355.71 (3.44)
	(64×32)	7.17 (0.08)	14.38 (0.25)	35.94 (0.39)	71.04 (0.99)	141.51 (2.04)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	0.32 (0.06)	0.81 (0.07)	4.62 (0.05)	13.10 (0.26)	36.79 (0.67)	206.79 (4.24)
	(60×4)	0.21 (0.04)	**0.54** (0.04)	**4.37** (0.05)	**12.64** (0.31)	**35.56** (0.79)	**184.60** (2.15)
	(60×8)	**0.17** (0.02)	0.59 (0.03)	4.54 (0.22)	16.47 (1.13)	39.35 (2.42)	205.34 (8.66)
	(60×16)	4.03 (1.50)	5.80 (1.05)	13.05 (0.46)	32.36 (0.68)	81.50 (1.96)	276.86 (3.60)
	(60×32)	1,780 (468)	7,004 (2,735)	2,983 (889)	1,967 (671)	4,037 (1,751)	3,209 (835)
HyCNN	(40×2)	0.24 (0.03)	0.98 (0.08)	5.56 (0.16)	13.51 (0.31)	35.06 (0.55)	193.32 (4.28)
	(40×4)	0.25 (0.04)	0.61 (0.05)	4.23 (0.08)	12.15 (0.28)	33.36 (0.79)	150.90 (2.61)
	(40×8)	0.21 (0.05)	0.61 (0.05)	4.13 (0.06)	**11.82** (0.42)	**32.65** (0.80)	115.02 (2.24)
	(40×16)	**0.13** (0.03)	**0.58** (0.04)	**4.05** (0.11)	12.35 (0.56)	34.90 (0.57)	**97.28** (1.43)
	(40×32)	0.14 (0.03)	0.63 (0.04)	4.20 (0.22)	15.67 (3.12)	83.10 (37.31)	1,039 (821)

Table 4: Prediction MSE (standard error) for different models and dimensions (**all entries multiplied by** 10^2), as in Table 2, but with true function $f_3(\mathbf{x}) = \|\mathbf{x}\|_2^2 + 0.25\sin(20\|\mathbf{x}\|_2)$ and $\sigma = 0$.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	0.32 (0.06)	2.54 (0.05)	3.62 (0.06)	6.45 (0.27)	27.06 (0.82)	359.01 (10.75)
	(84×4)	**0.08** (0.01)	**0.36** (0.10)	**1.07** (0.08)	**6.00** (0.12)	**24.01** (0.36)	**310.24** (7.42)
	(84×8)	0.24 (0.10)	0.49 (0.23)	1.25 (0.20)	6.22 (1.03)	25.77 (2.55)	318.23 (6.40)
	(84×16)	1.06 (0.26)	5.08 (2.54)	5.93 (4.15)	56.84 (13.39)	119.09 (23.76)	367.46 (31.12)
	(84×32)	10.89 (0.11)	21.42 (0.31)	47.31 (0.51)	90.59 (1.13)	177.62 (2.07)	442.82 (3.99)
ICNN	(64×2)	**2.74** (0.04)	**3.05** (0.04)	**3.60** (0.05)	**6.03** (0.08)	21.48 (0.49)	185.38 (3.04)
	(64×4)	4.14 (0.98)	6.53 (2.30)	3.77 (0.24)	14.18 (8.54)	**14.76** (0.14)	**155.52** (30.56)
	(64×8)	10.89 (0.11)	17.97 (2.40)	42.78 (4.14)	90.79 (1.14)	127.50 (23.20)	393.62 (35.87)
	(64×16)	10.89 (0.11)	21.45 (0.32)	47.39 (0.51)	90.75 (1.14)	178.51 (2.17)	446.64 (3.80)
	(64×32)	10.89 (0.11)	21.45 (0.32)	47.39 (0.51)	90.79 (1.14)	178.53 (2.17)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	2.86 (0.02)	3.03 (0.03)	3.41 (0.03)	4.99 (0.06)	17.77 (0.47)	162.67 (3.60)
	(60×4)	2.60 (0.04)	**2.91** (0.03)	**3.28** (0.05)	**4.65** (0.07)	**15.60** (0.41)	**135.96** (3.25)
	(60×8)	**2.58** (0.02)	2.91 (0.03)	3.41 (0.05)	5.89 (0.73)	17.40 (0.95)	182.08 (15.41)
	(60×16)	3.13 (0.13)	5.13 (0.47)	11.22 (1.26)	34.03 (0.82)	84.45 (1.33)	310.44 (3.16)
	(60×32)	166.77 (44.56)	1,260 (487)	1,043 (311)	1,098 (371)	3,016 (1,283)	3,210 (822)
HyCNN	(40×2)	2.85 (0.02)	3.12 (0.04)	3.59 (0.06)	6.14 (0.11)	19.16 (0.22)	149.87 (3.17)
	(40×4)	2.79 (0.02)	2.95 (0.02)	3.33 (0.04)	4.68 (0.05)	13.57 (0.32)	109.01 (1.53)
	(40×8)	2.72 (0.02)	2.95 (0.04)	3.30 (0.04)	4.11 (0.06)	9.06 (0.20)	99.14 (1.12)
	(40×16)	2.57 (0.02)	**2.90** (0.03)	**3.27** (0.04)	**3.89** (0.05)	**6.40** (0.07)	**72.19** (2.30)
	(40×32)	**2.54** (0.02)	2.95 (0.03)	3.39 (0.06)	5.79 (1.64)	32.53 (17.59)	751.70 (609.97)

Table 5: Prediction MSE (standard error) for different models and dimensions (**all entries multiplied by** 10^2), as in Table 2, but with true function $f_4(\mathbf{x}) = \|\mathbf{x}\|_1$.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	**0.25** (0.02)	**1.03** (0.09)	**12.01** (0.56)	**43.35** (2.11)	**101.20** (1.60)	503.41 (10.34)
	(84×4)	0.37 (0.05)	2.00 (0.12)	37.81 (1.87)	90.56 (2.45)	130.41 (2.85)	440.64 (8.46)
	(84×8)	0.41 (0.05)	1.74 (0.13)	30.91 (2.24)	80.75 (3.30)	129.32 (3.41)	433.41 (7.36)
	(84×16)	4.45 (1.24)	10.40 (2.37)	19.79 (4.42)	72.72 (6.33)	145.20 (8.88)	**404.26** (9.81)
	(84×32)	8.33 (0.05)	16.52 (0.18)	41.58 (0.37)	82.59 (1.03)	162.58 (1.65)	413.41 (4.01)
ICNN	(64×2)	**0.17** (0.04)	**0.52** (0.04)	**4.14** (0.11)	**13.52** (0.48)	41.54 (0.54)	242.05 (3.16)
	(64×4)	1.80 (1.03)	3.56 (1.97)	8.27 (3.60)	20.70 (6.78)	**39.70** (1.34)	**213.47** (21.38)
	(64×8)	8.34 (0.04)	15.02 (1.55)	37.80 (3.56)	82.86 (1.10)	132.37 (14.11)	386.25 (22.99)
	(64×16)	8.41 (0.07)	16.57 (0.19)	41.67 (0.35)	82.80 (1.10)	163.75 (1.74)	419.16 (3.81)
	(64×32)	8.34 (0.04)	16.55 (0.18)	41.67 (0.35)	82.82 (1.11)	163.75 (1.78)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	**0.09** (0.02)	**0.25** (0.02)	**1.83** (0.08)	**8.31** (0.17)	**35.15** (0.70)	219.43 (2.95)
	(60×4)	0.11 (0.03)	0.33 (0.02)	2.47 (0.11)	9.38 (0.23)	35.17 (0.61)	**198.67** (2.29)
	(60×8)	0.10 (0.03)	0.38 (0.03)	2.92 (0.29)	15.26 (1.85)	40.16 (3.27)	226.81 (11.09)
	(60×16)	3.08 (1.59)	5.20 (0.95)	15.72 (0.56)	38.74 (0.97)	94.05 (1.95)	321.00 (5.01)
	(60×32)	1,784 (471)	7,110 (2,768)	3,107 (933)	2,108 (720)	4,429 (1,922)	3,646 (950)
HyCNN	(40×2)	**0.10** (0.02)	**0.28** (0.02)	**2.44** (0.13)	**10.38** (0.25)	34.52 (0.31)	205.61 (3.43)
	(40×4)	0.11 (0.02)	0.40 (0.04)	2.89 (0.13)	10.88 (0.28)	33.78 (0.55)	155.20 (2.42)
	(40×8)	0.14 (0.03)	0.42 (0.04)	3.43 (0.12)	11.13 (0.37)	**31.29** (0.79)	117.15 (1.33)
	(40×16)	0.16 (0.03)	0.51 (0.04)	3.56 (0.08)	11.69 (0.46)	32.32 (0.45)	**98.84** (1.14)
	(40×32)	0.25 (0.04)	0.65 (0.05)	3.96 (0.27)	14.51 (2.72)	82.25 (38.33)	1,042 (820)

Table 6: Prediction MSE (standard error) for different models and dimensions (**all entries multiplied by** 10^2), as in Table 2, but with true function $f_5(\mathbf{x}) = \max(\|\mathbf{x} - \boldsymbol{\mu}\|^2, \|\mathbf{x} + \boldsymbol{\mu}\|^2)$ for a randomly drawn $\boldsymbol{\mu}$ with entries from $\mathcal{N}(0, 1/d)$ and $\sigma = 1$.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	**0.72** (0.23)	**1.33** (0.17)	**9.93** (0.62)	**35.06** (2.21)	**88.85** (2.08)	497.18 (10.63)
	(84×4)	0.92 (0.23)	2.34 (0.44)	31.04 (3.19)	73.41 (2.45)	107.24 (1.80)	**437.73** (10.81)
	(84×8)	0.75 (0.10)	2.38 (0.36)	19.66 (1.98)	62.25 (3.76)	112.50 (2.22)	447.43 (11.94)
	(84×16)	7.24 (4.92)	18.41 (8.41)	67.21 (21.99)	73.98 (21.01)	194.31 (24.40)	471.74 (21.53)
	(84×32)	75.49 (19.84)	98.37 (25.55)	131.74 (13.80)	164.71 (14.56)	247.54 (8.99)	509.05 (5.68)
ICNN	(64×2)	**0.27** (0.05)	**0.53** (0.05)	**2.89** (0.09)	**9.65** (0.31)	36.38 (0.81)	239.99 (4.83)
	(64×4)	22.66 (16.19)	54.22 (30.72)	4.52 (1.26)	36.48 (25.71)	**32.51** (1.50)	**206.81** (35.14)
	(64×8)	75.51 (19.85)	84.53 (26.75)	117.35 (16.28)	165.32 (14.59)	181.53 (29.16)	458.51 (37.20)
	(64×16)	75.56 (19.82)	98.48 (25.63)	131.81 (13.78)	165.09 (14.57)	249.28 (9.09)	515.78 (5.77)
	(64×32)	75.51 (19.85)	98.48 (25.63)	131.75 (13.77)	165.13 (14.57)	249.29 (9.12)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	0.18 (0.03)	0.53 (0.04)	2.56 (0.11)	8.53 (0.21)	29.84 (0.73)	213.91 (2.79)
	(60×4)	0.15 (0.04)	0.40 (0.03)	2.36 (0.08)	**7.89** (0.15)	**29.06** (0.92)	**190.90** (2.55)
	(60×8)	**0.13** (0.03)	**0.40** (0.02)	**2.27** (0.19)	10.99 (1.44)	31.22 (2.10)	226.07 (13.36)
	(60×16)	4.00 (1.38)	13.42 (2.85)	22.93 (3.30)	54.97 (1.88)	118.35 (3.06)	355.93 (4.60)
	(60×32)	3,000 (995)	13,606 (6,382)	5,186 (1,551)	2,972 (955)	5,751 (2,525)	4,331 (1,102)
HyCNN	(40×2)	0.16 (0.03)	0.60 (0.06)	2.90 (0.08)	8.70 (0.15)	29.21 (0.59)	202.84 (3.24)
	(40×4)	0.20 (0.04)	0.48 (0.03)	2.42 (0.06)	7.83 (0.28)	25.82 (0.62)	143.14 (2.81)
	(40×8)	0.21 (0.04)	0.50 (0.07)	2.39 (0.08)	7.71 (0.20)	**22.33** (0.86)	100.55 (1.26)
	(40×16)	**0.15** (0.03)	**0.41** (0.03)	**2.15** (0.08)	**7.06** (0.42)	22.61 (0.27)	**78.76** (1.17)
	(40×32)	0.24 (0.05)	0.47 (0.12)	2.48 (0.34)	11.02 (3.35)	83.79 (45.14)	1,179 (916)

Table 7: Prediction MSE (standard error) for different models and dimensions (**all entries multiplied by** 10^2), as in Table 2, but with true function $f_6(\mathbf{x}) = \exp(\|\mathbf{x}\|_1/\sqrt{d})$ with $\sigma = 1$.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	**0.37** (0.06)	**1.13** (0.12)	**11.99** (0.52)	**51.58** (1.51)	**253.32** (8.46)	10,253 (146)
	(84×4)	0.60 (0.11)	2.05 (0.16)	33.95 (1.22)	97.47 (2.86)	277.50 (6.67)	**9,432** (247)
	(84×8)	0.64 (0.13)	2.11 (0.15)	32.31 (2.49)	84.22 (4.34)	286.71 (12.79)	9,615 (316)
	(84×16)	0.98 (0.19)	13.11 (5.26)	65.21 (10.13)	151.23 (28.01)	574.49 (89.36)	10,587 (318)
	(84×32)	24.17 (0.16)	37.47 (0.49)	86.91 (1.10)	218.87 (3.38)	794.83 (10.01)	11,077 (136)
ICNN	(64×2)	**0.23** (0.05)	**0.60** (0.04)	**5.87** (0.25)	**27.53** (0.63)	151.46 (2.34)	5,412 (53)
	(64×4)	4.99 (3.00)	7.77 (4.53)	6.97 (0.83)	46.52 (19.13)	**127.54** (1.17)	**4,615** (657)
	(64×8)	24.18 (0.15)	34.15 (3.51)	78.38 (7.75)	219.32 (3.48)	590.20 (93.70)	10,412 (675)
	(64×16)	24.24 (0.15)	37.55 (0.50)	87.05 (1.09)	219.20 (3.48)	799.24 (10.20)	11,201 (123)
	(64×32)	24.18 (0.15)	37.51 (0.49)	87.05 (1.10)	219.29 (3.45)	799.24 (10.42)	$> 10^{11}$ ($\sim 10^{12}$)
GMN	(60×2)	0.17 (0.03)	0.45 (0.03)	**2.43** (0.11)	**11.68** (0.29)	148.44 (3.57)	5,216 (85)
	(60×4)	0.13 (0.04)	**0.41** (0.02)	2.84 (0.13)	13.33 (0.36)	**126.36** (2.46)	**4,460** (82)
	(60×8)	**0.10** (0.03)	0.43 (0.01)	3.51 (0.29)	20.98 (2.81)	135.15 (6.46)	5,639 (320)
	(60×16)	3.33 (1.26)	8.43 (1.58)	23.35 (0.90)	76.80 (1.94)	356.50 (6.27)	8,171 (93)
	(60×32)	1,991 (530)	8,301 (3,238)	4,122 (1,231)	3,754 (1,270)	15,390 (6,573)	80,717 (20,600)
HyCNN	(40×2)	0.15 (0.02)	0.61 (0.07)	4.54 (0.34)	24.43 (1.11)	169.18 (3.02)	4,883 (67)
	(40×4)	0.16 (0.02)	**0.49** (0.04)	**4.01** (0.18)	**22.38** (0.60)	127.19 (1.41)	3,634 (65)
	(40×8)	0.18 (0.04)	0.50 (0.05)	4.82 (0.26)	24.09 (0.41)	102.97 (1.30)	3,214 (64)
	(40×16)	**0.14** (0.03)	0.56 (0.03)	5.17 (0.19)	23.39 (0.82)	**92.33** (0.74)	**2,687** (74)
	(40×32)	0.20 (0.04)	0.65 (0.05)	5.70 (0.24)	33.67 (8.99)	390.89 (215.62)	21,663 (17,801)

D.2.5 Interpolation across different dimensions for sample size $n = 5{,}000$

We additionally consider the noiseless interpolation setting ($\sigma = 0$) for the same collection of functions (a, b, d-f) across different dimensions $d \in \{1, 2, 5, 10, 20, 50\}$ for a fixed sample size of $n = 5{,}000$. The noiseless results are reported in Table 8 (f_1), Table 9 (f_2), Table 10 (f_4), Table 11 (f_5), and Table 12 (f_6). For f_3 the prediction MSE is already reported in the noiseless setting in Table 4. The observations are consistent with the noisy setting, and we see that HyCNNs typically outperform all other architectures with the advantage being most pronounced in higher dimensions. Notably, in the noiseless setting, MLPs perform significantly better than ICNNs, which aligns with the theoretical considerations in Section 3, where we show that ReLU ICNNs suffer from limited expressivity due to a merely linear growth of linear pieces with depth.

Table 8: Prediction MSE and standard error (SE) by network and dimension (**all entries multiplied by** 10^2) for learning the function $f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2$ in the *noiseless* setting ($\sigma = 0$). All other settings are as in Table 2.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	$4.7 \cdot 10^{-3}$ ($2.7 \cdot 10^{-3}$)	0.02 (0.01)	**0.21** (0.02)	2.21 (0.04)	21.57 (0.48)	340.53 (8.42)
	(84×4)	$\mathbf{5.7 \cdot 10^{-5}}$ ($3.0 \cdot 10^{-5}$)	$\mathbf{3.3 \cdot 10^{-3}}$ ($7.1 \cdot 10^{-4}$)	0.29 (0.09)	2.49 (0.22)	**18.82** (0.44)	319.93 (6.16)
	(84×8)	0.01 (0.01)	0.01 ($1.4 \cdot 10^{-3}$)	0.27 (0.06)	**2.02** (0.14)	34.86 (14.92)	**294.75** (7.24)
	(84×16)	$3.8 \cdot 10^{-3}$ ($2.7 \cdot 10^{-3}$)	10.90 (2.81)	13.38 (6.29)	79.75 (8.28)	159.61 (14.89)	409.68 (20.72)
	(84×32)	8.87 (0.07)	17.71 (0.23)	44.38 (0.41)	88.10 (1.19)	174.46 (1.97)	440.22 (4.05)
ICNN	(64×2)	$\mathbf{1.8 \cdot 10^{-3}}$ ($2.4 \cdot 10^{-4}$)	**0.01** ($1.2 \cdot 10^{-3}$)	**0.27** (0.01)	**2.49** (0.05)	17.61 (0.32)	177.41 (2.44)
	(64×4)	1.77 (1.11)	3.48 (2.20)	0.43 (0.22)	10.67 (8.55)	**10.94** (0.18)	**150.86** (30.67)
	(64×8)	8.87 (0.07)	14.35 (2.28)	39.92 (4.19)	88.30 (1.20)	124.61 (23.13)	389.09 (36.85)
	(64×16)	8.87 (0.06)	17.73 (0.23)	44.46 (0.40)	88.27 (1.21)	175.34 (2.06)	443.96 (3.79)
	(64×32)	8.87 (0.07)	17.73 (0.23)	44.46 (0.40)	88.30 (1.20)	175.37 (2.07)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	$1.6 \cdot 10^{-3}$ ($6.7 \cdot 10^{-4}$)	0.01 ($8.6 \cdot 10^{-4}$)	0.12 (0.01)	1.44 (0.07)	14.40 (0.33)	157.64 (3.54)
	(60×4)	$\mathbf{1.2 \cdot 10^{-3}}$ ($4.0 \cdot 10^{-4}$)	$\mathbf{4.7 \cdot 10^{-3}}$ ($3.6 \cdot 10^{-4}$)	**0.09** (0.01)	**1.18** (0.07)	**11.98** (0.41)	**137.52** (2.91)
	(60×8)	$1.2 \cdot 10^{-3}$ ($1.5 \cdot 10^{-4}$)	0.01 ($3.7 \cdot 10^{-4}$)	0.12 (0.01)	2.47 (0.67)	13.01 (0.77)	177.07 (14.46)
	(60×16)	0.14 (0.08)	1.70 (0.42)	7.16 (1.27)	30.46 (0.92)	80.86 (1.24)	309.33 (4.02)
	(60×32)	133.50 (36.23)	1,044 (406)	968.37 (286.42)	1,059 (357)	2,970 (1,264)	3,190 (815)
HyCNN	(40×2)	$7.7 \cdot 10^{-4}$ ($1.8 \cdot 10^{-4}$)	0.01 ($2.9 \cdot 10^{-3}$)	0.27 (0.03)	2.62 (0.09)	16.34 (0.32)	147.29 (2.73)
	(40×4)	$7.1 \cdot 10^{-4}$ ($4.6 \cdot 10^{-4}$)	$2.9 \cdot 10^{-3}$ ($3.1 \cdot 10^{-4}$)	0.09 ($3.8 \cdot 10^{-3}$)	1.17 (0.02)	9.88 (0.22)	104.70 (1.30)
	(40×8)	$1.6 \cdot 10^{-4}$ ($3.4 \cdot 10^{-5}$)	$2.8 \cdot 10^{-3}$ ($7.2 \cdot 10^{-4}$)	0.04 ($1.7 \cdot 10^{-3}$)	0.60 (0.01)	5.51 (0.11)	94.59 (1.39)
	(40×16)	$\mathbf{1.2 \cdot 10^{-4}}$ ($2.1 \cdot 10^{-5}$)	$\mathbf{1.0 \cdot 10^{-3}}$ ($1.8 \cdot 10^{-4}$)	**0.03** ($4.6 \cdot 10^{-3}$)	**0.30** (0.01)	**2.94** (0.05)	**69.38** (1.79)
	(40×32)	$7.0 \cdot 10^{-4}$ ($2.3 \cdot 10^{-4}$)	0.01 ($2.3 \cdot 10^{-3}$)	0.13 (0.04)	2.25 (1.51)	29.48 (17.97)	772.69 (595.50)

Table 9: Prediction MSE and standard error (SE) by network and dimension (**all entries multiplied by** 10^2) for learning the function $f_2(\mathbf{x}) = \|\mathbf{x}\|_4^4$ in the *noiseless* setting ($\sigma = 0$), as in Table 8.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	$1.7 \cdot 10^{-3}$ $(7.5 \cdot 10^{-4})$	0.02 (0.01)	0.42 (0.02)	8.50 (0.25)	33.02 (0.61)	322.78 (8.01)
	(84×4)	0.01 (0.01)	0.03 (0.01)	0.38 (0.03)	9.16 (0.39)	34.36 (0.62)	300.74 (6.57)
	(84×8)	0.01 $(3.2 \cdot 10^{-3})$	0.03 (0.01)	0.39 (0.03)	8.72 (0.32)	44.73 (10.12)	299.67 (7.28)
	(84×16)	3.94 (1.05)	1.52 (1.37)	12.61 (4.89)	49.33 (8.93)	95.79 (17.42)	284.02 (18.91)
	(84×32)	7.13 (0.08)	14.34 (0.25)	35.77 (0.40)	70.64 (0.93)	140.46 (1.97)	351.73 (3.33)
ICNN	(64×2)	$1.5 \cdot 10^{-3}$ $(2.4 \cdot 10^{-4})$	0.01 $(8.4 \cdot 10^{-4})$	0.56 (0.06)	7.62 (0.11)	25.91 (0.31)	173.84 (3.04)
	(64×4)	1.40 (0.89)	2.85 (1.80)	1.30 (0.17)	14.20 (6.42)	20.97 (0.37)	144.42 (22.32)
	(64×8)	7.13 (0.08)	11.69 (1.86)	32.37 (3.25)	70.82 (0.95)	104.91 (16.77)	315.52 (25.65)
	(64×16)	7.13 (0.08)	14.36 (0.25)	35.86 (0.40)	70.80 (0.95)	141.06 (2.02)	354.30 (3.35)
	(64×32)	7.13 (0.08)	14.36 (0.25)	35.86 (0.40)	70.82 (0.95)	141.10 (2.02)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	$4.8 \cdot 10^{-3}$ $(1.9 \cdot 10^{-3})$	0.03 $(1.8 \cdot 10^{-3})$	0.52 (0.06)	7.65 (0.17)	24.54 (0.64)	161.02 (1.43)
	(60×4)	$4.1 \cdot 10^{-3}$ $(1.0 \cdot 10^{-3})$	0.01 $(1.2 \cdot 10^{-3})$	0.54 (0.10)	7.32 (0.10)	22.60 (0.30)	137.61 (1.67)
	(60×8)	0.01 $(4.4 \cdot 10^{-4})$	0.03 $(2.0 \cdot 10^{-3})$	0.90 (0.07)	8.48 (0.52)	23.34 (0.89)	172.69 (10.70)
	(60×16)	0.23 (0.11)	1.52 (0.36)	7.80 (0.90)	28.23 (0.59)	70.53 (0.98)	262.30 (3.39)
	(60×32)	109.42 (29.38)	838.73 (322.94)	781.01 (230.40)	852.66 (286.09)	2,379 (1,008)	2,561 (653)
HyCNN	(40×2)	$2.4 \cdot 10^{-3}$ $(4.9 \cdot 10^{-4})$	0.03 (0.01)	1.29 (0.14)	9.43 (0.16)	25.32 (0.47)	146.98 (1.64)
	(40×4)	$1.8 \cdot 10^{-3}$ $(8.1 \cdot 10^{-4})$	0.01 $(1.1 \cdot 10^{-3})$	0.67 (0.08)	7.19 (0.10)	20.61 (0.37)	112.98 (1.78)
	(40×8)	$5.5 \cdot 10^{-4}$ $(9.5 \cdot 10^{-5})$	0.01 $(2.3 \cdot 10^{-3})$	0.85 (0.08)	6.56 (0.13)	17.20 (0.23)	94.98 (0.86)
	(40×16)	$7.7 \cdot 10^{-4}$ $(1.5 \cdot 10^{-4})$	0.01 $(2.7 \cdot 10^{-3})$	1.57 (0.07)	6.31 (0.12)	15.51 (0.23)	78.81 (1.46)
	(40×32)	$5.3 \cdot 10^{-4}$ $(1.1 \cdot 10^{-4})$	0.03 (0.01)	2.03 (0.08)	8.14 (1.55)	36.81 (14.50)	654.80 (508.01)

Table 10: Prediction MSE and standard error (SE) by network and dimension (**all entries multiplied by** 10^2) for learning the function $f_4(\mathbf{x}) = \|\mathbf{x}\|_1$ in the *noiseless* setting ($\sigma = 0$), as in Table 8.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	$1.2 \cdot 10^{-3}$ $(7.6 \cdot 10^{-4})$	0.02 (0.01)	0.12 (0.04)	3.81 (0.39)	33.68 (0.76)	372.63 (10.65)
	(84×4)	$7.7 \cdot 10^{-4}$ $(5.5 \cdot 10^{-4})$	$3.8 \cdot 10^{-3}$ $(6.9 \cdot 10^{-4})$	0.39 (0.07)	8.52 (0.82)	33.40 (0.49)	323.00 (7.31)
	(84×8)	$1.3 \cdot 10^{-3}$ $(8.7 \cdot 10^{-4})$	0.07 (0.03)	0.24 (0.06)	7.55 (0.63)	32.92 (1.23)	309.03 (11.15)
	(84×16)	0.83 (0.78)	4.91 (2.37)	16.94 (6.34)	51.20 (11.15)	112.48 (18.65)	370.59 (23.49)
	(84×32)	8.31 (0.05)	16.50 (0.18)	41.56 (0.37)	82.58 (1.04)	162.55 (1.64)	413.43 (4.02)
ICNN	(64×2)	0.01 $(4.3 \cdot 10^{-4})$	0.07 (0.01)	0.34 (0.03)	5.70 (0.27)	27.83 (0.27)	192.76 (3.37)
	(64×4)	1.66 (1.05)	3.26 (2.03)	0.68 (0.31)	11.90 (7.68)	21.51 (0.26)	163.53 (26.01)
	(64×8)	8.31 (0.05)	14.96 (1.58)	37.41 (3.86)	82.74 (1.03)	119.97 (19.54)	373.88 (29.88)
	(64×16)	8.31 (0.05)	16.52 (0.18)	41.60 (0.37)	82.72 (1.04)	163.37 (1.70)	417.46 (3.76)
	(64×32)	8.31 (0.05)	16.52 (0.18)	41.61 (0.37)	82.74 (1.04)	163.39 (1.74)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	$7.6 \cdot 10^{-5}$ $(1.6 \cdot 10^{-5})$	$9.3 \cdot 10^{-4}$ $(1.2 \cdot 10^{-4})$	0.01 $(1.2 \cdot 10^{-3})$	0.09 (0.01)	20.72 (0.48)	175.21 (3.44)
	(60×4)	$1.9 \cdot 10^{-4}$ $(4.5 \cdot 10^{-5})$	$2.7 \cdot 10^{-3}$ $(7.0 \cdot 10^{-4})$	0.04 (0.01)	0.19 (0.03)	18.49 (0.44)	150.33 (2.36)
	(60×8)	$2.0 \cdot 10^{-4}$ $(6.7 \cdot 10^{-5})$	0.01 $(2.5 \cdot 10^{-3})$	0.12 (0.04)	2.33 (0.98)	18.90 (1.00)	192.45 (13.82)
	(60×16)	0.15 (0.07)	2.18 (0.44)	8.59 (1.35)	32.66 (0.93)	82.21 (1.32)	300.58 (2.62)
	(60×32)	124.21 (33.69)	978.13 (382.14)	904.45 (267.26)	988.83 (332.51)	2,791 (1,190)	2,999 (768)
HyCNN	(40×2)	$1.8 \cdot 10^{-5}$ $(5.3 \cdot 10^{-6})$	$3.4 \cdot 10^{-3}$ $(2.0 \cdot 10^{-3})$	0.03 $(4.6 \cdot 10^{-3})$	0.34 (0.04)	23.87 (0.41)	162.74 (1.86)
	(40×4)	$2.3 \cdot 10^{-4}$ $(1.6 \cdot 10^{-4})$	$1.2 \cdot 10^{-3}$ $(3.2 \cdot 10^{-4})$	0.03 $(4.8 \cdot 10^{-3})$	0.64 (0.04)	18.69 (0.10)	121.16 (1.88)
	(40×8)	$2.4 \cdot 10^{-4}$ $(9.2 \cdot 10^{-5})$	0.01 $(3.3 \cdot 10^{-3})$	0.04 $(4.0 \cdot 10^{-3})$	1.19 (0.03)	15.94 (0.21)	104.17 (1.24)
	(40×16)	$4.5 \cdot 10^{-3}$ $(1.1 \cdot 10^{-3})$	0.02 (0.01)	0.18 (0.03)	2.25 (0.15)	14.50 (0.16)	85.06 (1.81)
	(40×32)	0.02 $(2.2 \cdot 10^{-3})$	0.07 (0.01)	0.68 (0.15)	5.30 (1.32)	36.93 (15.52)	775.87 (572.90)

Table 11: Prediction MSE and standard error (SE) by network and dimension (**all entries multiplied by** 10^2) for learning the function $f_5(\mathbf{x}) = \max(\|\mathbf{x}-\boldsymbol{\mu}\|_2^2, \|\mathbf{x}+\boldsymbol{\mu}\|_2^2)$ in the *noiseless* setting ($\sigma = 0$) based on $n = 5{,}000$ data points $\mathbf{X}_i \sim \text{Unif}([-1,1]^d)$ and $Y_i = f_5(\mathbf{X}_i)$, where $\boldsymbol{\mu} \in \mathbb{R}^d$ has entries independently drawn from $\mathcal{N}(0, 1/d)$. All other settings are as in Table 2.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	0.04 (0.04)	**0.01** $(3.0 \cdot 10^{-3})$	0.35 (0.05)	**2.89** (0.12)	24.34 (0.50)	380.24 (10.93)
	(84×4)	**$2.5 \cdot 10^{-3}$** $(1.4 \cdot 10^{-3})$	0.02 (0.01)	**0.33** (0.03)	2.96 (0.20)	**24.08** (1.73)	330.84 (5.96)
	(84×8)	0.03 (0.01)	0.07 (0.03)	0.37 (0.07)	3.60 (0.78)	24.21 (0.75)	**323.04** (11.85)
	(84×16)	0.04 (0.01)	0.06 (0.01)	10.99 (9.48)	31.18 (26.19)	84.02 (30.31)	361.58 (40.33)
	(84×32)	75.45 (19.84)	98.33 (25.56)	131.70 (13.80)	164.71 (14.56)	247.55 (9.02)	509.07 (5.70)
ICNN	(64×2)	**0.04** (0.01)	**0.08** (0.02)	**0.54** (0.04)	**3.05** (0.11)	19.84 (0.37)	197.34 (3.25)
	(64×4)	22.42 (16.23)	46.00 (31.06)	1.23 (0.72)	29.86 (26.37)	**12.65** (0.33)	**169.03** (38.58)
	(64×8)	75.46 (19.85)	81.91 (27.46)	116.40 (16.85)	165.12 (14.57)	171.89 (33.37)	448.44 (42.94)
	(64×16)	75.46 (19.83)	98.45 (25.65)	131.77 (13.81)	164.98 (14.57)	248.85 (9.13)	514.14 (5.59)
	(64×32)	75.46 (19.84)	98.48 (25.67)	131.70 (13.79)	165.03 (14.56)	248.87 (9.15)	$> 10^{10}$ ($\sim 10^{10}$)
GMN	(60×2)	0.01 (0.01)	0.04 (0.01)	0.31 (0.04)	2.20 (0.12)	16.08 (0.38)	167.98 (3.53)
	(60×4)	**$2.5 \cdot 10^{-3}$** $(8.5 \cdot 10^{-4})$	**0.01** $(1.3 \cdot 10^{-3})$	**0.17** (0.01)	**1.63** (0.13)	**13.79** (0.43)	**149.01** (2.67)
	(60×8)	$3.2 \cdot 10^{-3}$ $(9.3 \cdot 10^{-4})$	0.02 $(4.5 \cdot 10^{-3})$	0.23 (0.03)	2.94 (0.74)	14.33 (0.76)	191.32 (16.92)
	(60×16)	0.41 (0.16)	8.20 (2.44)	14.35 (3.27)	45.05 (2.37)	104.09 (2.02)	343.18 (3.59)
	(60×32)	1,397 (631)	7,515 (4,700)	2,970 (909)	1,872 (581)	4,111 (1,791)	3,693 (925)
HyCNN	(40×2)	$4.5 \cdot 10^{-3}$ $(1.5 \cdot 10^{-3})$	0.07 (0.02)	0.61 (0.06)	3.06 (0.11)	17.23 (0.39)	161.93 (2.22)
	(40×4)	$4.1 \cdot 10^{-3}$ $(2.6 \cdot 10^{-3})$	0.02 (0.01)	0.20 (0.02)	1.43 (0.05)	11.10 (0.31)	114.28 (1.83)
	(40×8)	**$1.7 \cdot 10^{-3}$** $(5.2 \cdot 10^{-4})$	0.02 (0.01)	**0.10** (0.01)	0.77 (0.03)	6.24 (0.10)	100.81 (1.23)
	(40×16)	$4.4 \cdot 10^{-3}$ $(1.9 \cdot 10^{-3})$	**0.01** $(4.5 \cdot 10^{-3})$	0.11 (0.02)	**0.45** (0.01)	**3.75** (0.13)	**77.29** (2.43)
	(40×32)	0.04 (0.01)	0.13 (0.09)	0.40 (0.11)	3.25 (2.05)	37.74 (22.52)	803.29 (657.12)

Table 12: Prediction MSE and standard error (SE) by network and dimension (**all entries multiplied by** 10^2) for learning the function $f_6(\mathbf{x}) = \exp(\|\mathbf{x}\|_1/\sqrt{d})$ in the *noiseless* setting ($\sigma = 0$), as in Table 8.

	Network \ d	1	2	5	10	20	50
MLP	(84×2)	$3.5 \cdot 10^{-3}$ $(1.5 \cdot 10^{-3})$	**0.02** $(5.0 \cdot 10^{-3})$	**0.33** (0.06)	**15.70** (1.00)	186.32 (4.73)	10,099 (263)
	(84×4)	**$1.4 \cdot 10^{-3}$** $(7.5 \cdot 10^{-4})$	0.02 (0.01)	0.43 (0.04)	24.14 (0.72)	184.65 (5.77)	9,596 (203)
	(84×8)	$4.5 \cdot 10^{-3}$ $(2.8 \cdot 10^{-3})$	0.03 (0.01)	0.64 (0.11)	45.19 (19.28)	**177.00** (7.57)	**9,239** (304)
	(84×16)	0.01 $(4.0 \cdot 10^{-3})$	7.32 (4.57)	35.61 (12.93)	143.26 (30.61)	385.14 (89.22)	10,469 (429)
	(84×32)	24.14 (0.16)	37.44 (0.49)	86.88 (1.10)	218.86 (3.39)	794.79 (10.00)	11,077 (136)
ICNN	(64×2)	**0.01** $(9.4 \cdot 10^{-4})$	**0.05** $(4.4 \cdot 10^{-3})$	**0.44** (0.03)	**17.41** (0.41)	139.56 (2.25)	5,424 (88)
	(64×4)	4.80 (3.03)	7.34 (4.62)	0.93 (0.38)	34.75 (20.34)	**110.77** (1.11)	**4,608** (656)
	(64×8)	24.14 (0.16)	30.36 (4.79)	78.07 (8.00)	219.25 (3.39)	594.97 (91.47)	9,992 (843)
	(64×16)	24.15 (0.16)	37.51 (0.49)	87.02 (1.09)	219.21 (3.42)	799.00 (10.23)	11,198 (123)
	(64×32)	24.14 (0.16)	37.50 (0.49)	87.02 (1.09)	219.27 (3.38)	799.00 (10.44)	$> 10^{11}$ ($\sim 10^{12}$)
GMN	(60×2)	$2.3 \cdot 10^{-3}$ $(6.8 \cdot 10^{-4})$	0.02 $(1.6 \cdot 10^{-3})$	**0.10** (0.01)	**0.72** (0.08)	129.55 (2.10)	5,233 (101)
	(60×4)	$2.1 \cdot 10^{-3}$ $(4.5 \cdot 10^{-4})$	**0.01** $(1.8 \cdot 10^{-3})$	0.14 (0.02)	1.06 (0.08)	111.92 (2.32)	**4,405** (77)
	(60×8)	**$1.6 \cdot 10^{-3}$** $(2.5 \cdot 10^{-4})$	0.02 $(3.0 \cdot 10^{-3})$	0.32 (0.06)	5.59 (1.82)	**109.45** (3.76)	5,644 (350)
	(60×16)	0.33 (0.17)	3.57 (0.82)	13.27 (1.99)	68.30 (1.75)	344.22 (4.99)	8,138 (125)
	(60×32)	361.52 (98.19)	2,214 (860)	1,917 (564)	2,657 (896)	13,734 (5,833)	80,085 (20,435)
HyCNN	(40×2)	$2.3 \cdot 10^{-3}$ $(2.4 \cdot 10^{-4})$	0.03 (0.01)	0.20 (0.02)	5.62 (1.42)	158.98 (2.42)	4,968 (83)
	(40×4)	$1.7 \cdot 10^{-3}$ $(7.8 \cdot 10^{-4})$	**0.01** $(1.1 \cdot 10^{-3})$	**0.15** (0.02)	**3.54** (0.21)	110.24 (1.30)	3,715 (77)
	(40×8)	**$8.2 \cdot 10^{-4}$** $(1.7 \cdot 10^{-4})$	0.02 $(4.7 \cdot 10^{-3})$	0.20 (0.02)	6.63 (0.18)	87.98 (0.87)	3,237 (60)
	(40×16)	$1.5 \cdot 10^{-3}$ $(5.5 \cdot 10^{-4})$	0.03 (0.01)	0.61 (0.10)	9.55 (0.44)	**75.23** (1.32)	**2,707** (73)
	(40×32)	0.01 $(1.7 \cdot 10^{-3})$	0.10 (0.02)	1.64 (0.26)	20.91 (7.22)	305.51 (159.72)	21,612 (17,560)

D.2.6 Effect of the smoothing parameter on propagation and regression

We seek to understand the effect of the smoothing parameter $\tau \in \{0, 0.5, 1, 2, 5\}$ in (3) on the performance of the HyCNN of width $W = 40$ with input dimension $d = 50$, and depths $L \in \{2, 4, 6, 8, 10, 12, 14, 16\}$. Recall that for $\tau = 0$, the activation function $\overline{\sigma}_\tau$ is the maxout unit, whereas for $\tau > 0$ it is the log-sum-exp function.

Signal propagation and gradient profiles at initialization. To assess the effect on the signal and gradient profiles at initialization, we consider $5{,}000$ pairs $(\mathbf{X}_i, Y_i)$ of observations, with $\mathbf{X}_i \sim \mathcal{N}(0, I_{50})$ and $Y_i = \|\mathbf{X}_i\|_2^2 + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 1)$ are independent noise terms. We investigate the respective norm of the neuron activations across layers as well as the gradient norms with respect to the parameters under the least squares loss. To increase stability we standardize the covariates and responses first to have zero mean and unit variance across all dimensions. The gradients are depicted with respect to these standardized inputs and outputs. Moreover, for the hidden-to-hidden gradients we consider the unconstrained parameterization R_ℓ with $V_\ell = \mathrm{softplus}(R_\ell)$ where the softplus function is applied elementwise. This procedure is repeated 100 times across independent seeds for the data and the initialization weights. Figure 11 summarizes the resulting signal and gradient profiles at initialization.

Larger values of the smoothing parameter τ lead to more pronounced growth in both the signal and hidden-to-hidden gradient norms across layers at initialization. In particular, $\tau = 2$ and $\tau = 5$ exhibit relatively large amplification effects for large depth, which we attribute to the fact that the log-sum-exp exhibits a certain positive offset compared to the maxout unit and the initialization is currently tailored for the small $\tau = 0$ regime.

Signal propagation and gradient profiles after training. To exemplify the effect of the smoothing parameter in dimension $d = 50$ on the training and fit of the HyCNN, we consider the same regression tasks as in Appendix D.2.1 for $f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2$ and $f_4(\mathbf{x}) = \|\mathbf{x}\|_1$ with standard normal noise ($\sigma = 1$) and no noise ($\sigma = 0$), and train HyCNNs with depths $L \in \{2, 4, 6, 8, 10, 12, 14, 16\}$ and smoothing parameter $\tau \in \{0, 0.5, 1, 2, 5\}$. The results are shown in Figure 12. A small choice of the smoothing parameter τ leads to similar performance. In some instances, the HyCNN with $\tau = 0.5$ even outperforms the $\tau = 0$ HyCNN, however for $\tau \geq 1$ the performance consistently deteriorates with a more pronounced performance for depth $L \geq 10$. When considering the signal propagation and gradient norm plots in Figure 13 for the learned networks in regression task $f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2$ and noise $\sigma = 1$, we find that the hidden-to-hidden gradient norm is roughly one order of magnitude larger for deeper networks under $\tau = 5$ compared to the regime where $\tau = 0$. This indicates that deep HyCNNs with large τ were not properly trained in this regime. This behavior is at odds with some of the optimal transport simulations in Section D.3 where choosing τ large appears to help the training process under smooth ground truth OT map. We hypothesize that since the OT map estimator is given by the gradient of a HyCNN, choosing τ large serves as some benign gradient regularization, whereas for the regression tasks it does not help.

D.2.7 Effect of the number of activation branches

One of our core theoretical insights is that choosing a two-lane maxout activation yields a substantial gain of the expressive power of HyCNNs over ICNNs with a single-branch layout. This raises the question whether the use of further branches is beneficial.

To formalize the multi-branch setting, we define the K-branch HyCNN for $K \in \mathbb{N}, K \geq 2$ as

$$\mathrm{HyCNN}_K(\mathbf{x}) = \mathbf{z}_{L+1} = V_L \mathbf{z}_L + W_L \mathbf{x} + b_L, \quad \mathbf{z}_\ell = \max_{1 \leq k \leq K} \left(V_{\ell-1}^k \mathbf{z}_{\ell-1} + W_{\ell-1}^k \mathbf{x} + \mathbf{b}_{\ell-1}^k \right),$$

where $\mathbf{z}_0 = \mathbf{0}_d$, the maximum is taken coordinatewise, and V_ℓ^k are entrywise non-negative matrices.

To compare the performance of the multi-branch HyCNNs, we conduct experiments with $K = 2, 3, 4$ and varying depths $L \in \{2, 4, 6, 8, 10, 12, 14, 16\}$ for regression tasks with $f_1(\mathbf{x}) = \|\mathbf{x}\|_2^2$, $f_4(\mathbf{x}) = \|\mathbf{x}\|_1$, and $\sigma \in \{0, 1\}$. The width of the networks is chosen to lead to roughly comparable parameter counts across different depths, with widths 40, 30, and 24 for $K = 2, 3, 4$, respectively. We also consider $d = 50$ and $n = 5{,}000$ data points per repetition. For the initialization, we use the same initialization scheme as in Section 2. We consider training protocols analogous to those described in Appendix D.2.4 and the simulation results are shown in Figure 14.

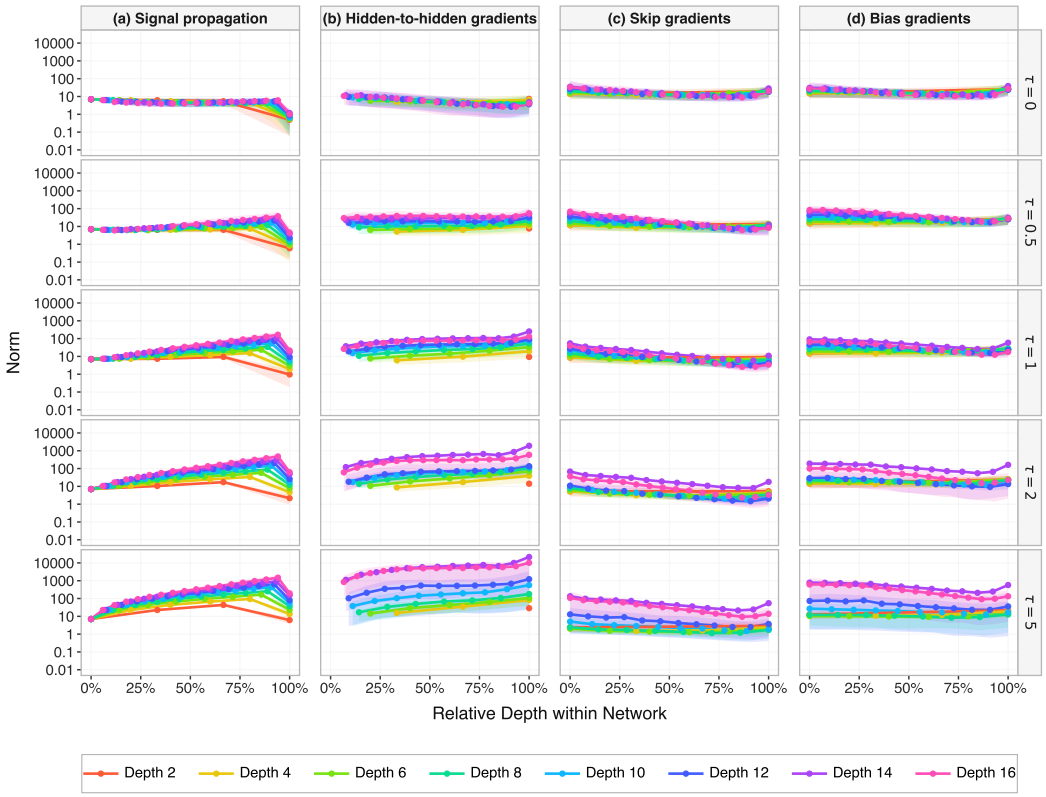

Figure 11: Signal propagation and gradient norm profiles across different layers at initialization of HyCNNs for $\tau = 0, 0.5, 1, 2, 5$ (top to bottom) and width 40 for different depths. The relative depth within network refers to the respective layer index normalized by network depth, with 0% referring to the input, and 100% to the scalar output of the network. **(a)** Signals are computed from a single forward pass at initialization with standard Gaussian input $\mathbf{X} \sim \mathcal{N}(0, I_d)$. Each curve is the mean hidden-state ℓ_2-norm of the neurons over 100 random seeds, and shaded bands show the 10th-90th percentile range. 0% corresponds to the input norm $\|\mathbf{X}\|_2$ (which is close to $\sqrt{d} \approx 7.07$) and 100% to the final hidden layer. **(b-d)** Gradients are computed from a single backward pass at initialization for least-squares regression on synthetic data of size $n = 5{,}000$, with $\mathbf{X}_i \sim \mathcal{N}(0, I_d)$ and $Y_i = \|\mathbf{X}_i\|_2^2 + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 1)$. Each curve is the mean of the gradient norm for (b) hidden-to-hidden weight, (c) skip weights, and (d) bias weights, and shaded bands show the 10th-90th percentile range.

The results indicate that increasing the number of activation branches beyond two does not lead to further improvements in prediction performance. Indeed, the two-branch HyCNN consistently achieves the lowest mean prediction MSE across all target functions, noise levels, and network depths considered in our experiments. It is worth noting that the initialization is tailored to the two-branch HyCNN, and no specific recalibration is performed for the three- or four-branch variants. This may contribute to the observed performance differences, particularly at larger depths. However, even at smaller depths, where the initialization mismatch should be less present, the two-branch model still outperforms the higher-branch variants, suggesting that the two-branch structure itself provides a favorable balance between expressivity and stable trainability.

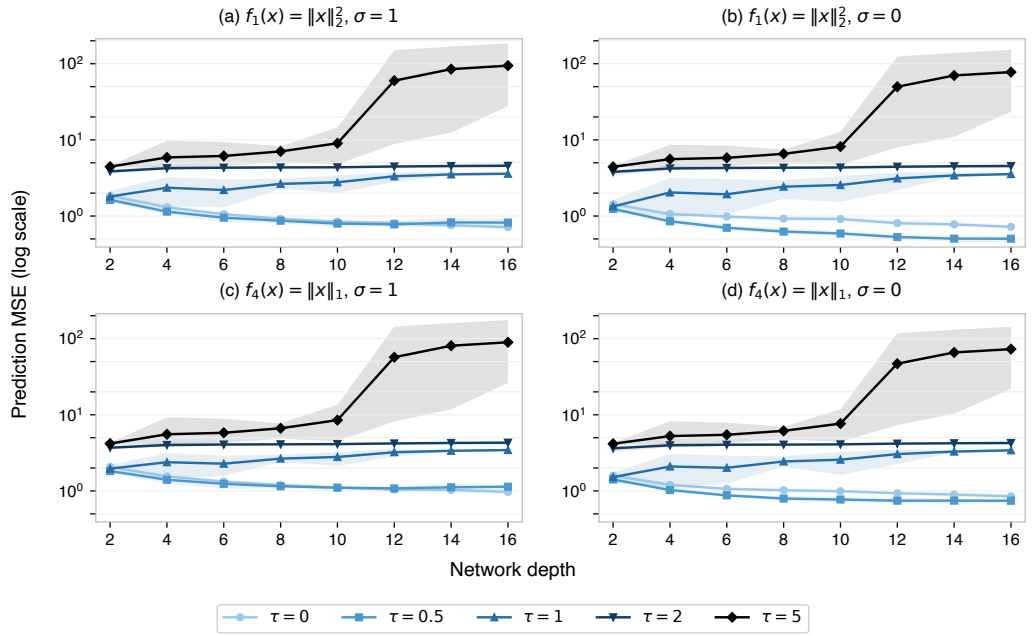

Figure 12: Prediction MSE of trained HyCNNs for width 40 and input dimension 50, for different depths and smoothing parameters $\tau \in \{0, 0.5, 1, 2, 5\}$. The HyCNNs are trained for 100 epochs with Adam based on 5,000 data points and batch size 1,000. As part of training the data is standardized to have mean zero and variance one across all dimensions, and the networks are afterward rescaled and evaluated on unstandardized inputs and targets. Shaded bands show the 10th–90th percentile range.

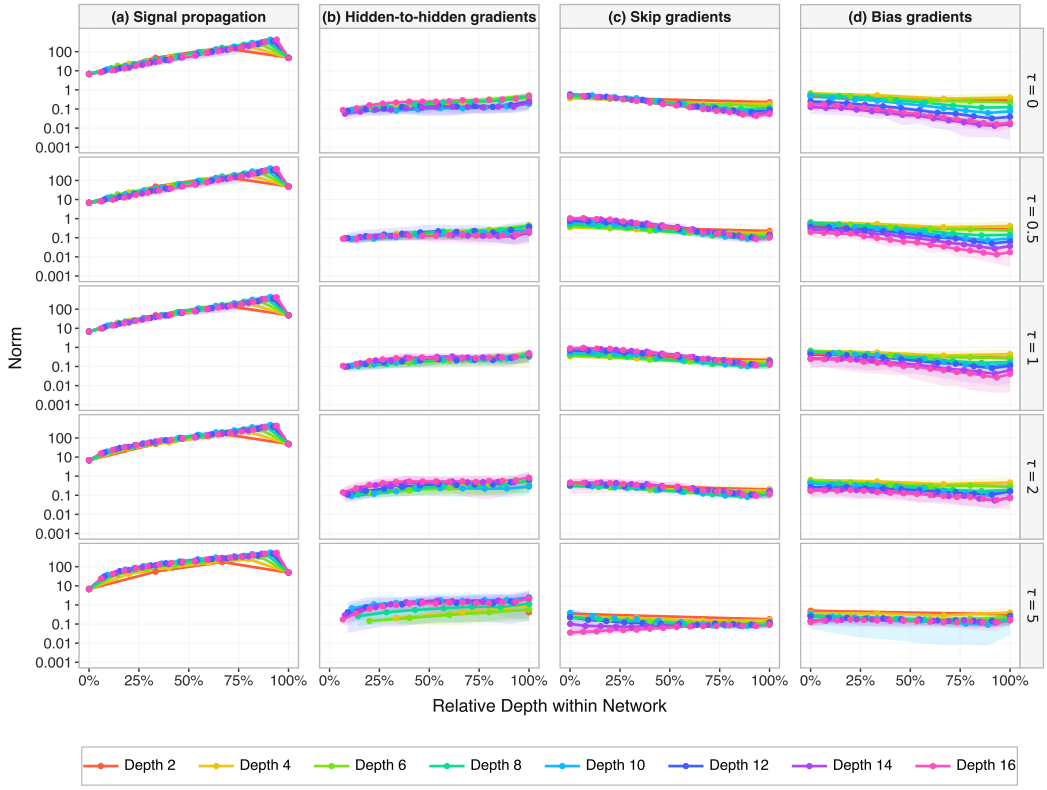

Figure 13: Signal propagation and gradient profiles after 100 epochs from the networks in Figure 12, for f_1 and $\sigma = 1$. Similar to the layout of Figure 11.

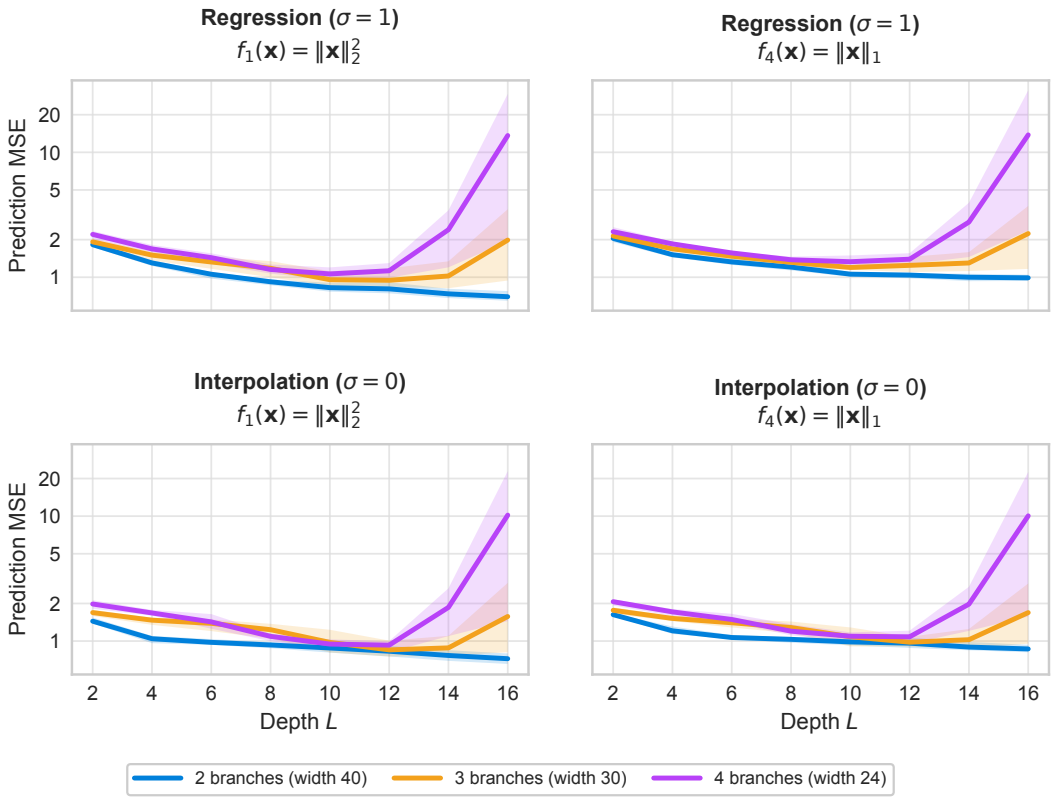

Figure 14: Performance of multi-branch HyCNNs for $K \in \{2, 3, 4\}$ activation branches for regression and interpolation tasks with $n = 5{,}000$ sample points in dimension $d = 50$. Curves show the prediction MSE across 10 independent runs, and the shaded regions depict the 10th–90th percentiles.

D.3 HyCNN-based OT map estimation

In this subsection, we provide additional experimental results and implementation details complementing the optimal transport map estimation experiments described in Section 5. The training procedure for HyCNN-based OT map estimation, based on the saddle-point formulation (6), is summarized in Algorithm 1.

As in the regression experiments of Section D.2, we enforce non-negativity of the hidden-to-hidden weight matrices in both the HyCNN potential f_{θ_f} and the HyCNN critic g_{θ_g} via the entrywise softplus reparametrization $V_\ell^k = \mathrm{softplus}(R_\ell^k) = \log(1 + \exp(R_\ell^k))$, where $R_\ell^k \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$ is unconstrained. The same reparametrization is used for the hidden-to-output weights. Adam updates the unconstrained matrices R_ℓ^k, with gradients propagated through the softplus function. Thus, the effective weights remain non-negative and both networks remain input-convex throughout training, without requiring projection onto the non-negative orthant.

Algorithm 1 HyCNN optimal transport map learning

Input: data $(\mathbf{x}_i)_{i=1}^n$ from source P, data $(\mathbf{y}_j)_{j=1}^m$ from target Q, batch size M, outer and inner loop iterations $S_{\text{out}}, S_{\text{in}} \in \mathbb{N}$, HyCNN potential $f_{\theta_f} \in \texttt{HyCNN}_\tau(\mathbb{R}^d)$ and HyCNN critic $g_{\theta_g} \in \texttt{HyCNN}_\tau(\mathbb{R}^d)$.
Initialize θ_f and θ_g according to Section 2.
for $t = 1$ **to** S_{out} **do**
  Sample mini-batch $(\mathbf{X}_i)_{i=1}^M$ from $(\mathbf{x}_i)_{i=1}^n$.
  for $s = 1$ **to** S_{in} **do**
    Sample mini-batch $(\mathbf{Y}_j)_{i=1}^M$ from $(\mathbf{y}_j)_{i=1}^m$.
    Update θ_g to maximize $\frac{1}{M}\sum_{i=1}^M \left[f_{\theta_f}(\mathbf{X}_i) + \langle \mathbf{Y}_i, \nabla g_{\theta_g}(\mathbf{Y}_i)\rangle - f_{\theta_f}(\nabla g_{\theta_g}(\mathbf{Y}_i)) \right]$ with Adam.
  end for
  Update θ_f to minimize $\frac{1}{M}\sum_{i=1}^M \left[f_{\theta_f}(\mathbf{X}_i) + \langle \mathbf{Y}_i, \nabla g_{\theta_g}(\mathbf{Y}_i)\rangle - f_{\theta_f}(\nabla g_{\theta_g}(\mathbf{Y}_i)) \right]$ using Adam.
end for
Output: Estimated optimal transport map $\widehat{T} := \nabla\widehat{\phi}$, where $\widehat{\phi} = f_{\theta_f}$.

Network architectures. Both the potential f_{θ_f} and the critic g_{θ_g} are parameterized by HyCNNs with the smooth log-sum-exp activation function $\overline{\sigma}_\tau$ from (3), using the same architecture (width and depth).

Training. For all experiments, we use the Adam optimizer (Kingma & Ba, 2015) with $\beta_1 = 0.5$ and $\beta_2 = 0.9$ and identical initial learning rate $\lambda = 10^{-2}$ for both θ_f and θ_g. The learning rate follows a cosine decay schedule $\lambda_t = \lambda_{\min} + \frac{1}{2}(\lambda - \lambda_{\min})(1 + \cos(\pi t / S_{\text{out}}))$ with final ratio $\lambda_{\min}/\lambda = 0.01$. Non-negativity of the effective weights in both networks is maintained by the softplus reparametrization described above, with Adam updating the corresponding unconstrained parameters.

Notably, unlike the ICNN-based neural OT method by Makkuva et al. (2020), we do not allow the hidden-to-hidden weight matrices V_ℓ^j in the critic network to have negative entries, as we found that it degrades performance of HyCNN-based OT map estimation in high-dimensional settings.

Training in low dimensions. While discontinuities of the OT map can occur in any dimension, in low dimensions it is crucial to accurately resolve these discontinuities, as they have a pronounced effect on the quality of the estimated map. This requires the HyCNN to operate close to the maxout unit (i.e., $\tau \to 0$ in the log-sum-exp activation $\overline{\sigma}_\tau$). However, training with small τ is challenging because the maxout unit may select a single gate per layer, so that only the weights associated with the active gate receive gradient updates while the remaining gates are not trained. To address this, we employ a cyclic cosine annealing schedule for the smoothing parameter τ, that is, we start from $\tau_0 = 1$ and then periodically oscillate τ between large and small values and decrease per cycle by a fixed decay factor. During phases of large τ, the smooth log-sum-exp activation distributes gradients across all gates, ensuring that all weight matrices are updated. During phases of small τ, the network sharpens toward the piecewise-affine $\max$ regime needed to represent discontinuous maps.

Training in moderate and high dimensions. In higher dimensions ($d \geq 10$), recovering the OT map is already more involved, and empirically larger values of τ yielded better performance. We therefore use a fixed smoothing parameter τ without the cyclic cosine schedule in these settings.

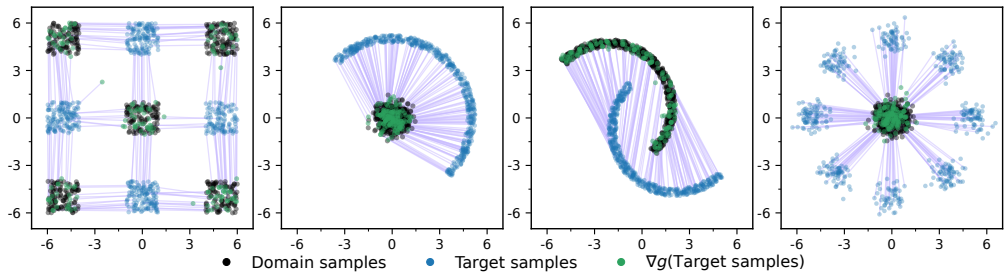

Figure 15: Reverse OT map estimation in dimension $d = 2$. Each panel shows the pushforward of the target distribution under the learned gradient of the learned HyCNN g (width 40, depth 4) from the inner optimization problem in Algorithm 1, trained with $n = m = M = 2{,}000$ data points, $S_{\text{out}} = 2500$ outer iterations, and $S_{\text{in}} = 10$ inner steps, using Adam with cosine-decayed learning rate ($\lambda = 10^{-2}$, final ratio 0.01) and smoothing parameter $\tau_0 = 1$ (cosine decay). From left to right: (i) five-component checkerboard → four-component checkerboard, (ii) standard normal → half-moon, (iii) shifted half-moon → shifted rotated half-moon, (iv) standard Gaussian → mixture of eight Gaussians.

D.3.1 OT map estimation in dimension $d = 2$

For the two-dimensional OT visualization experiments, we consider the following examples:

(i) five-component checkerboard (source) to four-component checkerboard (target),

(ii) standard normal (source) to half-moon (target),

(iii) shifted half-moon (source) to shifted rotated half-moon (target), and

(iv) standard Gaussian (source) to a mixture of eight Gaussians arranged in a ring (target).

In Figure 3 we depict the training samples from the source distribution and target distribution as well as the pushforward of the source training samples under the learned HyCNN OT map ∇f. In Figure 15 we show the pushforward of the target training samples under the learned gradient of the critic g, which corresponds to the reverse OT map. In Figure 16, we visualize the learned potentials f and g for the four examples using a contour plot. Throughout these examples, we observe that HyCNNs are capable of learning the correct target distribution (and thus the correct OT map) even in the presence of discontinuities.

For the architecture we use HyCNNs with width 40 and depth 4 for both the potential f and the critic g. We draw $n = m = M = 2{,}000$ data points from the source and target distributions. The training consists of $S_{\text{out}} = 2500$ outer iterations for learning f, with $S_{\text{in}} = 10$ inner gradient steps per outer iteration for updating the critic g. The learning rate follows a cosine decay schedule with initial learning rate $\lambda = 10^{-2}$ and final ratio $\lambda_{\min}/\lambda = 0.01$. The smoothing parameter τ for the log-sum-exp activation is also decayed. We start with $\tau = 1$, and within a cycle of 100 outer iterations, we decay the smoothing parameter τ by a factor of 0.8 per cycle, with a cosine decaying schedule and ensure that by iteration 70% of the cycle, τ has decayed to 0.1 times its initial value in the cycle. Overall, this ensures that τ reaches a small value of $0.8^{25} \cdot 0.1 = 0.00038$ by the end of training, which allows the network to sharpen toward the piecewise-affine $\max$ regime needed to represent discontinuous maps, while also ensuring that all gates receive gradient updates during training.

D.3.2 OT map estimation across different dimensions for sample sizes $n = m = 5{,}000$

In the following, we present detailed performance tables for the OT map estimation experiments described in Section 5. In total, we consider the following OT potentials with corresponding OT

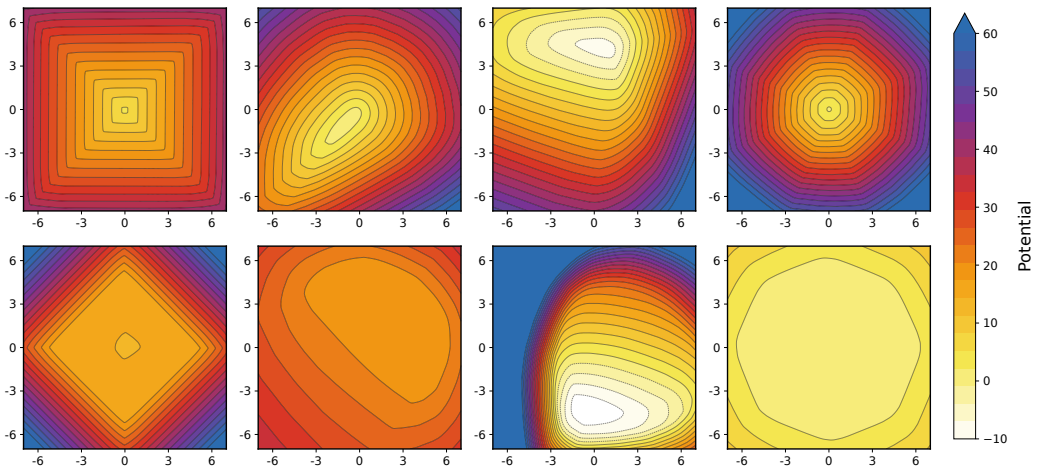

Figure 16: Contour plots of the learned OT potential $\widehat{\phi}$ in dimension $d = 2$. Each panel shows the contour lines of the HyCNN potential (width 40, depth 4) learned from $n = m = M = 2{,}000$ data points for the same collection of source–target pairs as in Figure 15.

maps,

$$\begin{aligned}
&\text{(a) } \phi_1(\mathbf{x}) = \tfrac{\|\mathbf{x}\|_2^2}{2}, && T_1(\mathbf{x}) = \mathbf{x},\\
&\text{(b) } \phi_2(\mathbf{x}) = \tfrac{1}{2}\textstyle\sum_{i=1}^{d}\big(1 + \tfrac{\sin(i)}{2}\big)x_i^2, && T_2(\mathbf{x}) = \big((1 + \tfrac{\sin(i)}{2})x_i\big)_{i\in[d]},\\
&\text{(c) } \phi_3(\mathbf{x}) = \tfrac{\|\mathbf{x}\|_2^2}{2} + \|\mathbf{x}\|_1, && T_3(\mathbf{x}) = \big(x_i + \operatorname{sign}(x_i)\big)_{i\in[d]},\\
&\text{(d) } \phi_4(\mathbf{x}) = \|\mathbf{x}\|_4^4, && T_4(\mathbf{x}) = \big(4x_i^3\big)_{i\in[d]}.
\end{aligned}$$

we report the prediction MSE of the estimated OT map $\widehat{T} = \nabla\widehat{\phi}$ against the ground-truth OT map $\overline{T} = \nabla\phi_\ell$ for dimension $d = 50$. The source distribution is $P = \mathcal{N}(0, I_d)$ for (a–c) and $P = \mathrm{Unif}[-1,1]^d$ for (d), and the target distribution is $Q = (\nabla\phi_\ell)_{\#}P$. We use $n = m = 5{,}000$ independent samples from P and Q, respectively. The MSE is defined as the average squared Euclidean distance between the estimated and ground-truth OT map evaluated on an independent test set, i.e.,

$$\mathrm{MSE}_{\mathrm{pred}} = \frac{1}{N_{\mathrm{test}}}\sum_{i=1}^{N_{\mathrm{test}}} \|\widehat{T}(\mathbf{X}_i) - \overline{T}(\mathbf{X}_i)\|_2^2.$$

The test set consists of $N_{\mathrm{test}} = 1{,}000$ independent samples from P and Q. Results are averaged over 10 independent runs with different random seeds for sample generation and network initialization. The training uses batch size $M = 256$, $S_{\mathrm{out}} = 1{,}000$ outer iterations, $S_{\mathrm{in}} = 5$ inner steps, learning rate $\lambda = 10^{-2}$ with cosine decay with final ratio 0.01. The Adam optimizer uses $\beta_1 = 0.5$, $\beta_2 = 0.9$ for both the potential and critic networks.

We compare the HyCNN-based estimator (width 44, depths $L \in \{2, 4, 6\}$, log-sum-exp smoothness $\tau \in \{0.1, 1, 10\}$) with the entropic OT map estimator ($\epsilon \in \{0.1, 1, 10\}$), the Monge gap estimator (MLP width 104, depths $L \in \{2, 4, 6\}$), and multiple ICNN variants (width 64, depths $L \in \{2, 4, 6\}$). Every configuration is evaluated over ten repetitions and the widths of the neural networks are chosen to ensure comparable numbers of parameters, see Table 13. In the following, we describe how these other estimators are implemented.

Entropic OT map estimator. As a first nonparametric baseline we consider the entropic OT (EOT) map estimator (Seguy et al., 2018; Pooladian & Niles-Weed, 2021). Given independent samples $(\mathbf{X}_i)_{i=1}^n \sim P$ and $(\mathbf{Y}_j)_{j=1}^m \sim Q$, let $\widehat{P}_n = \frac{1}{n}\sum_i \delta_{\mathbf{X}_i}$ and $\widehat{Q}_m = \frac{1}{m}\sum_j \delta_{\mathbf{Y}_j}$ denote the empirical measures. The estimator involves solving the entropic OT problem with squared-Euclidean cost

$c(\mathbf{x},\mathbf{y}) = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2$ and regularization parameter $\varepsilon > 0$,

$$\min_{\pi \in \Pi(\widehat{P}_n, \widehat{Q}_m)} \sum_{i,j} \pi_{ij}\, c(\mathbf{X}_i, \mathbf{Y}_j) + \varepsilon\, \mathrm{KL}\big(\pi \,\big\|\, \tfrac{1}{n}\mathbf{1} \otimes \tfrac{1}{m}\mathbf{1}\big). \tag{42}$$

For our purposes, we will consider $\varepsilon \in \{0.1, 1, 10\}$. The above optimization problem can be solved using the Sinkhorn algorithm (Cuturi, 2013; Peyré & Cuturi, 2019). In particular, as part of the optimization procedure, the dual entropic OT problem is solved,

$$\max_{f \in \mathbb{R}^n, g \in \mathbb{R}^m} \sum_{i=1}^n f_i + \sum_{j=1}^m g_j - \varepsilon \sum_{i,j} \exp\left(\frac{f_i + g_j - c(\mathbf{X}_i, \mathbf{Y}_j)}{\varepsilon}\right),$$

where all dual optimizers $(f^\star, g^\star)$ are characterized by

$$f_i^\star = -\varepsilon \log \sum_{j=1}^m \exp\left(\frac{g_j^\star - c(\mathbf{X}_i, \mathbf{Y}_j)}{\varepsilon}\right), \qquad i \in [n],$$
$$g_j^\star = -\varepsilon \log \sum_{i=1}^n \exp\left(\frac{f_i^\star - c(\mathbf{X}_i, \mathbf{Y}_j)}{\varepsilon}\right), \qquad j \in [m].$$

From the dual potential $g^\star$ associated with $\widehat{Q}_m$, the entropic OT map is then defined according to Pooladian & Niles-Weed (2021) as

$$\widehat{T}_\varepsilon(\mathbf{x}) = \sum_{j=1}^m w_j^\varepsilon(\mathbf{x})\, \mathbf{Y}_j, \qquad w_j^\varepsilon(\mathbf{x}) = \frac{\exp\big(\big(g_j^\star - c(\mathbf{x}, \mathbf{Y}_j)\big)/\varepsilon\big)}{\sum_{j'=1}^m \exp\big(\big(g_{j'}^\star - c(\mathbf{x}, \mathbf{Y}_{j'})\big)/\varepsilon\big)},$$

which coincides with the barycentric map (i.e., the conditional expectation of $\mathbf{Y}$ given $\mathbf{X} = \mathbf{x}$) induced by the entropic OT plan from (42) on the training samples $\{\mathbf{X}_i\}_{i=1}^n$. In our numerical simulations, we rely on the `geomloss` package (Feydy et al., 2019) using the tensorized backend, squared-Euclidean cost, blur parameter $\sqrt{\varepsilon}$, and no debiasing to compute $g^\star$ which is then used to construct the entropic OT map $\widehat{T}_\varepsilon$ as described above.

Monge gap MLP estimator. As a further baseline that does not rely on input-convex parameterizations, we consider the Monge gap MLP estimator of Uscidda & Cuturi (2023). Here the OT map is parameterized directly as a plain MLP $T_\theta : \mathbb{R}^d \to \mathbb{R}^d$ with ELU activations (following the original implementation) and hidden widths $(104, \dots, 104)$ of depth $L \in \{2, 4, 6\}$ (the same candidate depths as the ICNN baselines). For source and target mini-batches $\mathbf{X}_1, \dots, \mathbf{X}_M$ and $\mathbf{Y}_1, \dots, \mathbf{Y}_M$ of size M, let $\widehat{P}_M = \frac{1}{M}\sum_i \delta_{\mathbf{X}_i}$, $\widehat{Q}_M = \frac{1}{M}\sum_i \delta_{\mathbf{Y}_i}$ and $(T_\theta)_\# \widehat{P}_M = \frac{1}{M}\sum_i \delta_{T_\theta(\mathbf{X}_i)}$ denote the associated empirical measures. Upon denoting the entropic OT cost for $c(\mathbf{x},\mathbf{y}) = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$ by

$$\mathrm{OT}_\varepsilon(\mu, \nu) := \min_{\pi \in \Pi(\mu,\nu)} \int c\, \mathrm{d}\pi + \varepsilon\, \mathrm{KL}(\pi \,\|\, \mu \otimes \nu),$$

the training objective combines a (debiased) Sinkhorn divergence

$$\mathcal{S}_\varepsilon(\mu, \nu) = \mathrm{OT}_\varepsilon(\mu, \nu) - \tfrac{1}{2}\mathrm{OT}_\varepsilon(\mu, \mu) - \tfrac{1}{2}\mathrm{OT}_\varepsilon(\nu, \nu)$$

under squared-Euclidean costs with the entropic Monge gap regularizer of Uscidda & Cuturi (2023), namely

$$\min_\theta \underbrace{\mathcal{S}_\varepsilon\big((T_\theta)_\# \widehat{P}_M, \widehat{Q}_M\big)}_{\text{fitting loss}} + \lambda \underbrace{\Big[\tfrac{1}{M}\textstyle\sum_{i=1}^M c\big(\mathbf{X}_i, T_\theta(\mathbf{X}_i)\big) - \mathrm{OT}_\varepsilon\big(\widehat{P}_M, (T_\theta)_\# \widehat{P}_M\big)\Big]}_{\text{Monge gap } \mathcal{M}_\varepsilon(T_\theta; \widehat{P}_M)}. \tag{43}$$

The (unregularized) Monge gap $\mathcal{M}_0(T_\theta; \widehat{P}_M) \geq 0$ vanishes whenever T_θ transports $\widehat{P}_M$ to $(T_\theta)_\# \widehat{P}_M$ optimally, i.e., whenever T_θ is itself nearly the gradient of a convex function, and choosing ϵ small ensures that the Monge gap is computationally efficiently computable while being close to $\mathcal{M}_0$. Both terms in (43) are evaluated via differentiable Sinkhorn iterations provided by the `ott-jax` library of Cuturi et al. (2022). We set the entropic regularization to $\varepsilon = 0.05$, the Monge gap weight to $\lambda = 0.1$, and use mean cost normalization (i.e., normalizing the cost function along the training samples to ensure that the average cost equals one) following the default configuration in Uscidda & Cuturi (2023). Training minimizes (43) by Adam with fixed learning rate 10^{-2} as the objective

involves no saddle point problem, batch size $M = 256$, and $S_{\text{out}} = 1{,}000$ outer iterations; all remaining hyperparameters are kept at their `ott-jax` defaults.

ICNN-based OT map estimators. In addition, we compare the HyCNN-based OT map estimator to four ICNN variants, which we collectively refer to as *ICNN-based methods*. All variants share the general ICNN template of (1), i.e., hidden activations

$$\mathbf{z}_{\ell+1} = \sigma_\ell\big(V_\ell \mathbf{z}_\ell + W_\ell \mathbf{x} + \mathbf{b}_\ell\big), \qquad \ell = 0, \dots, L-1,$$

with scalar output $\mathrm{ICNN}(\mathbf{x}) = V_L \mathbf{z}_L + W_L \mathbf{x} + \mathbf{b}_L$, and differ only in their first-layer parameterization and choice of activation σ_ℓ.

- (i) *ICNN (ReLU).* The standard ICNN with first layer $\mathbf{z}_1 = \sigma(W_0 \mathbf{x} + \mathbf{b}_0)$ and ReLU activation function $\sigma(a) = \max(a, 0)$ in every layer.
- (ii) *ICNN (Leaky ReLU).* Identical to (i) but with the leaky-ReLU activation $\sigma(x) = \max(x, \alpha x)$, where $\alpha = 0.2$ is the negative slope.
- (iii) *ICNNq (ReLU).* An ICNN whose first hidden layer is augmented by a learned quadratic skip connection:
$$\mathbf{z}_1 = \sigma\big(W_0 \mathbf{x} + (W_{\mathrm{q}} \mathbf{x})^{\odot 2} + \mathbf{b}_0\big), \tag{44}$$
where $W_{\mathrm{q}} \in \mathbb{R}^{d_1 \times d}$ is a learned weight matrix, $(\,\cdot\,)^{\odot 2}$ denotes elementwise squaring, and σ is the ReLU activation; all subsequent layers are defined as in the standard ICNN. The quadratic term injects a non-negative, input-convex contribution into the first layer and improves the ability of ICNNs to represent mildly curved potentials.
- (iv) *ICNNq (Softplus).* Identical to (iii) but with the activation replaced by the smooth Softplus $\sigma_\tau(a) = \tau \log(1 + e^{a/\tau})$ with smoothness $\tau > 0$, which recovers ReLU as $\tau \to 0$. We consider parameter choices $\tau \in \{0.1, 1, 10\}$ and report the results for all three values.

The potential f_{θ_f} and the critic g_{θ_g} of each ICNN baseline are both instantiated as ICNNs of the same variant (i)–(iv), width, and depth, but they are treated differently regarding the non-negativity of their hidden-to-hidden weight matrices V_ℓ. We enforce the potential f_{θ_f} to be input-convex by utilizing for $V_\ell \geq 0$ the same softplus reparametrization used for the HyCNN: each V_ℓ is represented as the entrywise softplus $V_\ell = \mathrm{softplus}(R_\ell) = \log(1 + \exp(R_\ell))$ of an unconstrained parameter tensor R_ℓ, so that $V_\ell \in (0, \infty)^{n_{\ell+1} \times n_\ell}$ holds automatically on every forward pass and Adam can update R_ℓ in $\mathbb{R}^{n_{\ell+1} \times n_\ell}$ without any projection. For the critic g_{θ_g}, by contrast, strict input-convexity is not needed: g_{θ_g} enters the saddle problem only through the gradient $\nabla g_{\theta_g}(\mathbf{Y})$ at target samples, and a mild soft convexity suffices to make the inner Legendre-type problem well-posed. We therefore drop the softplus reparametrization for g_{θ_g} and let each hidden-to-hidden matrix of g_{θ_g} be an ordinary unconstrained real parameter, with non-negativity merely encouraged softly through the convexity penalty $\lambda_{\mathrm{cvx}} \|(V_\ell)_-\|_F^2$ that appears in the training objective (45) below.

At initialization, all V_ℓ are drawn so that their first and second moments match the values derived by Hoedt & Klambauer (2023) for ICNNs, namely

$$\mu_W = \sqrt{\tfrac{6\pi}{n_\ell D_\ell}}, \qquad \sigma_W^2 = \tfrac{1}{n_\ell}, \qquad D_\ell = 6(\pi - 1) + (n_\ell - 1)\big(3\sqrt{3} + 2\pi - 6\big),$$

where n_ℓ is the fan-in of layer ℓ. For the critic g_{θ_g}, whose hidden-to-hidden matrices are unconstrained, we draw $(V_\ell)_{ij} \sim \mathcal{N}(\mu_W, \sigma_W^2)$ directly. For the potential f_{θ_f}, whose effective weight must be non-negative by construction, we instead draw V_ℓ entrywise from the log-normal distribution with the same prescribed moments, $(V_\ell)_{ij} \sim \mathrm{LogNormal}(\mu, \sigma^2)$ with

$$\mu = \log\Big(\tfrac{\mu_W^2}{\sqrt{\mu_W^2 + \sigma_W^2}}\Big), \quad \sigma^2 = \log\Big(1 + \tfrac{\sigma_W^2}{\mu_W^2}\Big),$$

and then store the corresponding pre-image $R_\ell = \mathrm{softplus}^{-1}(V_\ell) = \log(\exp(V_\ell) - 1)$. By construction the effective weight at initialization is always positive and satisfies $\mathbb{E}[(V_\ell)_{ij}] = \mu_W$ and $\mathrm{Var}[(V_\ell)_{ij}] = \sigma_W^2$. We choose the log-normal distribution (rather than a truncated Gaussian) because it is inherently non-negative and allows us to match the prescribed mean and variance of Hoedt & Klambauer (2023) simultaneously.

The remaining parameters are identical for f_{θ_f} and g_{θ_g}: the input-to-hidden matrices W_ℓ are unconstrained and initialized entrywise as $(W_\ell)_{ij} \sim \mathcal{N}(0, 1/n_\ell)$ (LeCun-normal), the hidden-layer biases

are set to the constant $\mu_b = -\sqrt{3n_\ell/D_\ell}$ and the output bias to zero, and for the two quadratic-skip variants (iii)–(iv) the additional first-layer matrix $W_{\mathrm{q}} \in \mathbb{R}^{n_1 \times d}$ is drawn entrywise as $\mathcal{N}(0, 1/d)$. All random matrices are drawn independently across layers.

All ICNN-based methods are trained by solving an empirical saddle-point problem similar to our HyCNN-based estimator, with the HyCNN potential and critic replaced by the respective ICNN variant and an additional regularization term. We refer to the resulting objective as the ICNN objective; for mini-batches $(\mathbf{X}_i)_{i\in[M]}$ and $(\mathbf{Y}_j)_{j\in[M]}$ of size M, it takes the form

$$\min_{\theta_f} \max_{\theta_g} \left\{ \frac{1}{M} \sum_{i=1}^{M} \Big[f_{\theta_f}(\mathbf{X}_i) + \langle \mathbf{Y}_i, \nabla g_{\theta_g}(\mathbf{Y}_i) \rangle - f_{\theta_f}(\nabla g_{\theta_g}(\mathbf{Y}_i)) \Big] \; - \; \lambda_{\mathrm{cvx}} \sum_{V \in \theta_g} \|(V)_-\|_F^2 \right\}, \tag{45}$$

where $(V)_- := \max(0, -V)$ denotes the negative part of V taken elementwise, $\|\cdot\|_F$ is the Frobenius norm, and the sum ranges over all hidden-to-hidden weight matrices V of the critic network g_{θ_g}. The parameter $\lambda_{\mathrm{cvx}} > 0$ controls the strength of the convexity regularization and softly penalizes negative entries in the hidden-to-hidden weights of the critic but does not enforce strict non-negativity. We use the same value $\lambda_{\mathrm{cvx}} = 1$ for all ICNN-based methods. The potential f_{θ_f} and the critic g_{θ_g} are parameterized by the same architecture (width, depth, activation).

For training we follow the same alternating optimization scheme as for the HyCNN-based estimator, which is inspired by Makkuva et al. (2020, Algorithm 1). We run $S_{\mathrm{out}} = 1{,}000$ outer iterations with $S_{\mathrm{in}} = 5$ inner steps, batch size $M = 256$, and the Adam optimizer (Kingma & Ba, 2015) with $\beta_1 = 0.5$, $\beta_2 = 0.9$, initial learning rate $\lambda = 10^{-2}$, and cosine decay to final ratio 0.01, applied identically to both θ_f and θ_g. The OT map estimate is then read off as $\widehat{T} = \nabla f_{\theta_f}$.

Prediction MSE tables for OT map estimation. We first present the prediction MSE of the estimated OT map $\widehat{T} = \nabla\widehat{\phi}$ against the ground-truth OT map $\overline{T} = \nabla\phi_\ell$ for each of the four OT potentials (a–d) and dimensions $d \in \{10, 20, 50\}$. The MSE is evaluated on an independent test set of $N_{\mathrm{test}} = 1{,}000$ samples; mean and standard error (SE) are computed over 10 independent runs. We compare a comprehensive set of estimators: ICNN with ReLU and Leaky ReLU activations, ICNNq with a quadratic skip connection and either ReLU or Softplus activations (the latter for varying smoothing parameters $\tau \in \{0.1, 1, 10\}$), HyCNN with the log-sum-exp activation for $\tau \in \{0.1, 1, 10\}$, the Monge gap estimator, and entropic OT (EOT) with entropy regularization parameter $\epsilon \in \{0.1, 1, 10\}$. We only report the results for the HyCNN architecture (depth L and smoothness τ) that achieves the smallest Sinkhorn divergence on an independent validation set as described in the next section, and we report the results for all τ variants of ICNNs to provide a comprehensive picture of the performance of these baselines across different smoothness levels. The Monge gap MLP estimator uses an MLP with width 104, while the ICNN architectures use width 64 and the HyCNNs use width 44, each with depths $L \in \{2, 4, 6\}$.

The results in Tables 14, 16, 18, and 20 reveal several consistent patterns. On the smooth quadratic targets (a) and (b), HyCNN is the best estimator in every dimension and the three HyCNN architectures form the three smallest MSE entries throughout, dominating all baselines, including the strongest ICNN-based competitor, ICNNq (Softplus), by a factor of roughly 3–5 in MSE. Here the large smoothness setting $\tau = 10$ is uniformly preferred by the Sinkhorn validation criterion. On the non-smooth target (c), HyCNN remains best at $d \in \{20, 50\}$ and is only marginally surpassed by the tuned ICNNq (Softplus) configuration at $d = 10$. On the strongly curved target (d), by contrast, ICNNq (Softplus) with a small to moderate temperature $\tau \in \{0.1, 1\}$ clearly beats HyCNN at $d = 10$ and $d = 20$, whereas HyCNN regains the lead at $d = 50$. For HyCNN on (c) and (d) the optimal smoothness is target- and dimension-dependent, with small $\tau \in \{0.1, 1\}$ preferred at $d = 10$ and $\tau = 10$ most accurate and stable at $d = 50$. Among the ICNN baselines, the quadratic-skip variants (ICNNq) clearly outperform the plain ICNN variants across all settings, and the Softplus activation is consistently more accurate and more stable than ReLU once τ is tuned: small $\tau \in \{0.1, 1\}$ is preferred at $d \in \{10, 20\}$, while large $\tau = 10$ is required at $d = 50$ to avoid the depth-induced instabilities exhibited by ICNNq (ReLU) at $L \in \{4, 6\}$ on targets (c) and (d). Even ICNNq (Softplus) with small $\tau = 0.1$ degrades sharply at $L = 6$, $d = 50$ on the non-smooth and strongly curved targets, confirming that low smoothness amplifies high-dimensional instabilities. ICNN (L-ReLU) is the least robust method overall and exhibits catastrophic failures at $L = 2$, $d = 50$ across all four targets. The best entropic OT estimator is uniformly worst on (a)–(d), and the Monge gap is outperformed by HyCNN by an order of magnitude in nearly all settings. HyCNN remains stable across $L \in \{2, 4, 6\}$ once τ is

chosen appropriately, with its relative advantage most pronounced on the smooth quadratic targets and at high dimensions.

Sinkhorn divergence based network selection.

In addition to the prediction MSE tables above, we present Sinkhorn divergence tables (with regularization $\varepsilon = 0.1$) evaluated on an independent validation set of $n_{\text{valid}} = 1{,}000$ sample pairs from P and $Q = (\nabla\phi_\ell)_{\#}P$, computed separately from the training and test data. This validation metric assesses the distributional fit of the learned OT map and serves as a guide for selecting the HyCNN architecture (depth L and smoothing parameter τ) for a given problem. Concretely, for a HyCNN estimator $\widehat{T}$ based on training data, we compute for empirical measures $\widehat{P}_{n_{\text{valid}}}, \widehat{Q}_{n_{\text{valid}}}$ the Sinkhorn divergence (Feydy et al., 2019; Peyré & Cuturi, 2019) based on the squared Euclidean distance with regularization parameter $\epsilon = 0.1$,

$$\mathcal{S}_\epsilon(\widehat{T}_{\#}\widehat{P}_{n_{\text{valid}}}, \widehat{Q}_{n_{\text{valid}}}).$$

For each simulation run, we draw a single pair of validation data sets from domain and target and compute the above Sinkhorn divergence. Given the 10 independent runs for training, we thus compute 10 independent Sinkhorn divergence values which we use to compute mean and standard error (SE). The resulting validation metrics for the HyCNN-based OT estimators are depicted in Tables 15, 17, 19, and 21, right below the respective prediction MSE table with the same ground truth.

The Sinkhorn divergence validation metrics demonstrate that a well-performing HyCNN estimator can be consistently identified through our validation procedure. We conducted analogous experiments with alternative choices of the regularization parameter ϵ and found the validation technique to yield consistent results across these settings. A noteworthy observation is that identifying an estimator with even a moderately smaller Sinkhorn divergence can translate into a significantly improved fit. We find this phenomenon striking and attribute it to high-dimensional effects as it is inherently difficult to certify a substantially better distributional fit on the basis of independent realizations alone.

Table 13: Total neural-network parameter counts for the OT estimators at the selected widths, including weights and biases. For the MLP semidual estimator, ICNN variants, and HyCNN, totals include both scalar-valued networks, the potential f and critic g. The Monge gap MLP estimator uses a single d-dimensional map network. Network size is $(m \times L)$, with L hidden layers; the semidual MLP has width 72, the Monge gap MLP has width 104, the ICNN variants have width 64, and HyCNN has width 44 at every depth and dimension. Counts are identical across the four OT tasks and all displayed activation and smoothness settings. Entropic OT is nonparametric and has no neural-network parameters; its data-dependent Sinkhorn dual values are excluded.

Method	$(m \times L)$	$d = 10$	$d = 20$	$d = 50$
Monge gap (MLP)	(104×2)	13,114	15,204	21,474
	(104×4)	34,954	37,044	43,314
	(104×6)	56,794	58,884	65,154
MLP (semidual)	(72×2)	12,242	13,682	18,002
	(72×4)	33,266	34,706	39,026
	(72×6)	54,290	55,730	60,050
ICNN (semidual)	(64×2)	11,158	13,738	21,478
	(64×4)	30,358	35,498	50,918
	(64×6)	49,558	57,258	80,358
ICNNq (semidual)	(64×2)	12,438	16,298	27,878
	(64×4)	31,638	38,058	57,318
	(64×6)	50,838	59,818	86,758
HyCNN (semidual)	(44×2)	11,726	15,266	25,886
	(44×4)	31,086	38,146	59,326
	(44×6)	50,446	61,026	92,766

Table 14: Prediction MSE (mean and SE over 10 runs) of the estimated OT map for the OT map (a) $T_1(\mathbf{x}) = \mathbf{x}$ based on $n = m = 5{,}000$ independent training sample points from the source and target distribution. Per HyCNN architecture ($44 \times L$), the τ variant with the smallest Sinkhorn divergence validation metric ($\epsilon = 0.1$) for 1,000 validation samples at that dimension is used; see Table 15. Per dimension, the smallest mean MSE per method type is bold and colored, with the three smallest values across all methods underlined. Monge gap uses MLP width 104, ICNN variants use width 64, and both HyCNN networks use width 44.

Method \ d		10	20	50
Entropic OT	$\epsilon = 0.1$	2.678 (0.017)	12.098 (0.039)	51.783 (0.075)
	$\epsilon = 1$	**0.999** (0.008)	**6.537** (0.034)	41.019 (0.067)
	$\epsilon = 10$	6.538 (0.022)	13.129 (0.031)	**32.786** (0.058)
Monge gap	(104×2)	**1.167** (0.025)	**4.586** (0.049)	23.385 (0.131)
	(104×4)	1.275 (0.039)	5.416 (0.084)	23.007 (0.105)
	(104×6)	1.262 (0.059)	5.793 (0.101)	**21.038** (0.141)
ICNN (ReLU)	(64×2)	0.852 (0.021)	3.221 (0.061)	20.792 (0.567)
	(64×4)	0.498 (0.010)	2.222 (0.038)	12.205 (0.352)
	(64×6)	**0.391** (0.007)	**1.965** (0.065)	**9.809** (0.186)
ICNN (L-ReLU)	(64×2)	1.213 (0.055)	4.360 (0.135)	614.316 (216.829)
	(64×4)	0.734 (0.029)	2.599 (0.089)	20.861 (1.983)
	(64×6)	**0.679** (0.130)	**2.002** (0.072)	**10.183** (0.474)
ICNNq (ReLU)	(64×2)	**0.222** (0.009)	**0.801** (0.067)	**4.101** (0.299)
	(64×4)	0.286 (0.011)	1.123 (0.062)	9.581 (0.479)
	(64×6)	3.788 (3.404)	1.401 (0.049)	127.802 (79.678)
ICNNq (Softplus)	$(64 \times 2); \tau = 0.1$	0.121 (0.008)	0.567 (0.031)	3.350 (0.143)
	$(64 \times 2); \tau = 1$	**0.073** (0.002)	**0.365** (0.020)	3.032 (0.256)
	$(64 \times 2); \tau = 10$	0.195 (0.007)	0.486 (0.010)	**2.046** (0.028)
	$(64 \times 4); \tau = 0.1$	0.192 (0.004)	0.858 (0.036)	7.563 (0.283)
	$(64 \times 4); \tau = 1$	0.103 (0.003)	0.407 (0.017)	6.950 (0.244)
	$(64 \times 4); \tau = 10$	0.188 (0.008)	0.490 (0.007)	2.367 (0.020)
	$(64 \times 6); \tau = 0.1$	0.233 (0.010)	0.971 (0.010)	10.011 (1.030)
	$(64 \times 6); \tau = 1$	0.097 (0.003)	0.570 (0.050)	6.617 (1.388)
	$(64 \times 6); \tau = 10$	0.186 (0.007)	0.504 (0.005)	2.482 (0.021)
HyCNN	(44×2)	0.028 (0.001)	0.109 (0.003)	0.783 (0.019)
	(44×4)	**0.023** (0.002)	**0.074** (0.003)	0.405 (0.007)
	(44×6)	0.063 (0.002)	0.079 (0.003)	**0.381** (0.009)

Table 15: Sinkhorn divergence (mean and SE over 10 runs) of the estimated OT map for the OT map (a) $T_1(\mathbf{x}) = \mathbf{x}$. Per HyCNN architecture ($44 \times L$) and per dimension, the τ variant with the smallest mean Sinkhorn divergence is bold and colored, with the three smallest values across all HyCNN rows underlined.

Method \ d		10	20	50
HyCNN	$(44 \times 2); \tau = 0.1$	4.324 (0.027)	16.055 (0.055)	61.958 (0.198)
	$(44 \times 2); \tau = 1$	4.116 (0.022)	15.671 (0.050)	60.970 (0.088)
	$(44 \times 2); \tau = 10$	**4.037** (0.021)	**14.907** (0.033)	**57.797** (0.098)
	$(44 \times 4); \tau = 0.1$	4.201 (0.024)	15.640 (0.035)	62.698 (0.099)
	$(44 \times 4); \tau = 1$	4.063 (0.020)	15.338 (0.041)	60.437 (0.081)
	$(44 \times 4); \tau = 10$	**4.047** (0.021)	**14.900** (0.033)	**57.566** (0.108)
	$(44 \times 6); \tau = 0.1$	4.228 (0.028)	15.598 (0.042)	61.734 (0.143)
	$(44 \times 6); \tau = 1$	**4.042** (0.021)	15.056 (0.031)	59.548 (0.148)
	$(44 \times 6); \tau = 10$	4.045 (0.020)	**14.896** (0.031)	**57.533** (0.088)

Table 16: Prediction MSE (mean and SE over 10 runs) of the estimated OT map for the OT map (b) $T_2(\mathbf{x}) = \left((1 + \frac{\sin(i)}{2})\, x_i\right)_{i\in[d]}$ based on $n = m = 5{,}000$ independent training sample points from the source and target distribution. Per HyCNN architecture $(44 \times L)$, the τ variant with the smallest Sinkhorn divergence validation metric ($\epsilon = 0.1$) for 1,000 validation samples at that dimension is used; see Table 17. Per dimension, the smallest mean MSE per method type is bold and colored, with the three smallest values across all methods underlined.

Method \ d		10	20	50
Entropic OT	$\epsilon = 0.1$	2.927 (0.018)	12.404 (0.033)	52.669 (0.107)
	$\epsilon = 1$	**1.078** (0.011)	**6.589** (0.034)	41.518 (0.101)
	$\epsilon = 10$	7.435 (0.026)	14.022 (0.040)	**33.704** (0.084)
Monge gap	(104×2)	**1.461** (0.029)	5.239 (0.062)	22.050 (0.144)
	(104×4)	1.631 (0.048)	5.419 (0.099)	19.884 (0.196)
	(104×6)	1.707 (0.030)	**5.175** (0.069)	**17.915** (0.120)
ICNN (ReLU)	(64×2)	1.010 (0.010)	3.822 (0.058)	19.617 (0.202)
	(64×4)	0.597 (0.007)	2.341 (0.064)	12.451 (0.180)
	(64×6)	**0.426** (0.007)	**1.939** (0.066)	**9.786** (0.248)
ICNN (L-ReLU)	(64×2)	1.389 (0.071)	4.702 (0.157)	60.328 (17.675)
	(64×4)	0.779 (0.031)	2.735 (0.153)	16.110 (0.854)
	(64×6)	**0.541** (0.021)	**2.161** (0.068)	**9.996** (0.275)
ICNNq (ReLU)	(64×2)	**0.249** (0.020)	**0.694** (0.046)	**3.633** (0.241)
	(64×4)	0.318 (0.029)	1.059 (0.037)	8.347 (0.260)
	(64×6)	0.368 (0.018)	1.455 (0.060)	26.901 (9.868)
ICNNq (Softplus)	$(64 \times 2); \tau = 0.1$	0.139 (0.008)	0.550 (0.023)	3.259 (0.174)
	$(64 \times 2); \tau = 1$	**0.078** (0.003)	**0.324** (0.012)	3.447 (0.257)
	$(64 \times 2); \tau = 10$	0.242 (0.013)	0.570 (0.006)	**2.019** (0.056)
	$(64 \times 4); \tau = 0.1$	0.215 (0.008)	0.833 (0.031)	7.303 (0.289)
	$(64 \times 4); \tau = 1$	0.108 (0.003)	0.445 (0.037)	6.358 (0.318)
	$(64 \times 4); \tau = 10$	0.242 (0.007)	0.553 (0.011)	2.419 (0.034)
	$(64 \times 6); \tau = 0.1$	0.251 (0.012)	1.022 (0.019)	20.532 (12.168)
	$(64 \times 6); \tau = 1$	0.111 (0.003)	0.603 (0.028)	4.063 (0.280)
	$(64 \times 6); \tau = 10$	0.243 (0.009)	0.584 (0.015)	2.472 (0.024)
HyCNN	(44×2)	0.030 (0.001)	0.109 (0.002)	0.710 (0.017)
	(44×4)	**0.024** (0.002)	**0.078** (0.004)	**0.413** (0.005)
	(44×6)	0.032 (0.004)	0.100 (0.004)	0.478 (0.007)

Table 17: Sinkhorn divergence (mean and SE over 10 runs) of the estimated OT map for the OT map (b) $T_2(\mathbf{x}) = \left((1 + \frac{\sin(i)}{2})\, x_i\right)_{i\in[d]}$. Per HyCNN architecture $(44 \times L)$ and per dimension, the τ variant with the smallest mean Sinkhorn divergence is bold and colored, with the three smallest values across all HyCNN rows underlined.

Method \ d		10	20	50
HyCNN	$(44 \times 2); \tau = 0.1$	4.571 (0.025)	16.288 (0.070)	61.705 (0.245)
	$(44 \times 2); \tau = 1$	4.333 (0.025)	15.608 (0.042)	61.781 (0.092)
	$(44 \times 2); \tau = 10$	**4.257** (0.024)	**14.911** (0.035)	**58.321** (0.112)
	$(44 \times 4); \tau = 0.1$	4.455 (0.029)	15.683 (0.050)	62.971 (0.217)
	$(44 \times 4); \tau = 1$	4.293 (0.027)	15.302 (0.037)	60.082 (0.235)
	$(44 \times 4); \tau = 10$	**4.276** (0.023)	**14.911** (0.037)	**58.102** (0.113)
	$(44 \times 6); \tau = 0.1$	4.463 (0.030)	15.604 (0.038)	62.175 (0.105)
	$(44 \times 6); \tau = 1$	4.277 (0.029)	15.047 (0.030)	59.842 (0.260)
	$(44 \times 6); \tau = 10$	**4.269** (0.027)	**14.919** (0.043)	**58.444** (0.123)

Table 18: Prediction MSE (mean and SE over 10 runs) of the estimated OT map for the OT map (c) $T_3(\mathbf{x}) = (x_i + \text{sign}(x_i))_{i\in[d]}$ based on $n = m = 5{,}000$ independent training sample points from the source and target distribution. Per HyCNN architecture $(44 \times L)$, the τ variant with the smallest Sinkhorn divergence validation metric ($\epsilon = 0.1$) for 1,000 validation samples at that dimension is used; see Table 19. Per dimension, the smallest mean MSE per method type is bold and colored, with the three smallest values across all methods underlined.

Method \ d		10	20	50
Entropic OT	$\epsilon = 0.1$	8.155 (0.033)	42.243 (0.072)	189.406 (0.327)
	$\epsilon = 1$	**3.834** (0.024)	**30.997** (0.111)	169.342 (0.385)
	$\epsilon = 10$	16.868 (0.025)	33.891 (0.029)	**89.193** (0.108)
Monge gap	(104×2)	**5.666** (0.068)	**19.595** (0.046)	87.219 (0.238)
	(104×4)	8.067 (0.225)	19.795 (0.087)	76.500 (0.365)
	(104×6)	8.612 (0.133)	21.680 (0.222)	**63.571** (0.475)
ICNN (ReLU)	(64×2)	4.670 (0.263)	20.552 (0.149)	89.585 (1.698)
	(64×4)	**2.165** (0.133)	13.822 (0.240)	66.244 (2.288)
	(64×6)	2.461 (0.209)	**9.641** (0.673)	**55.948** (0.771)
ICNN (L-ReLU)	(64×2)	134.056 (115.902)	2.29e+03 (1.57e+03)	3.82e+03 (1.13e+03)
	(64×4)	2.784 (0.196)	13.704 (0.131)	85.385 (2.898)
	(64×6)	**2.271** (0.194)	**11.384** (0.232)	**59.944** (2.004)
ICNNq (ReLU)	(64×2)	1.522 (0.067)	8.499 (0.057)	**26.985** (0.571)
	(64×4)	**1.220** (0.026)	7.211 (0.350)	48.320 (1.869)
	(64×6)	1.558 (0.093)	**5.828** (0.387)	61.714 (0.967)
ICNNq (Softplus)	$(64 \times 2); \tau = 0.1$	1.605 (0.165)	8.410 (0.055)	28.272 (1.198)
	$(64 \times 2); \tau = 1$	3.519 (0.035)	8.172 (0.025)	**24.779** (0.495)
	$(64 \times 2); \tau = 10$	4.769 (0.036)	9.492 (0.040)	25.550 (0.229)
	$(64 \times 4); \tau = 0.1$	**0.942** (0.021)	6.896 (0.494)	44.097 (2.595)
	$(64 \times 4); \tau = 1$	3.231 (0.071)	8.967 (0.105)	30.605 (0.596)
	$(64 \times 4); \tau = 10$	4.620 (0.047)	9.634 (0.068)	27.468 (0.149)
	$(64 \times 6); \tau = 0.1$	1.069 (0.018)	**5.025** (0.480)	228.345 (118.208)
	$(64 \times 6); \tau = 1$	2.878 (0.081)	9.241 (0.102)	28.474 (0.415)
	$(64 \times 6); \tau = 10$	4.565 (0.047)	9.568 (0.056)	27.948 (0.100)
HyCNN	(44×2)	1.917 (0.055)	7.731 (0.023)	21.750 (0.120)
	(44×4)	1.052 (0.010)	**4.986** (0.176)	19.872 (0.040)
	(44×6)	**1.027** (0.009)	5.275 (0.189)	**19.763** (0.047)

Table 19: Sinkhorn divergence (mean and SE over 10 runs) of the estimated OT map for the OT map (c) $T_3(\mathbf{x}) = (x_i + \text{sign}(x_i))_{i\in[d]}$. Per HyCNN architecture $(44 \times L)$ and per dimension, the τ variant with the smallest mean Sinkhorn divergence is bold and colored, with the three smallest values across all HyCNN rows underlined.

Method \ d		10	20	50
HyCNN	$(44 \times 2); \tau = 0.1$	**10.741** (0.092)	53.760 (0.268)	222.568 (0.837)
	$(44 \times 2); \tau = 1$	11.836 (0.122)	54.513 (0.182)	209.047 (0.371)
	$(44 \times 2); \tau = 10$	14.598 (0.076)	**53.144** (0.078)	**206.854** (0.282)
	$(44 \times 4); \tau = 0.1$	**10.558** (0.076)	50.522 (0.124)	220.174 (0.273)
	$(44 \times 4); \tau = 1$	11.177 (0.078)	**49.013** (0.261)	211.551 (0.211)
	$(44 \times 4); \tau = 10$	14.652 (0.085)	53.057 (0.073)	**205.994** (0.357)
	$(44 \times 6); \tau = 0.1$	**10.582** (0.074)	**50.479** (0.145)	215.187 (0.456)
	$(44 \times 6); \tau = 1$	11.457 (0.075)	50.502 (0.179)	208.538 (0.310)
	$(44 \times 6); \tau = 10$	14.651 (0.077)	53.021 (0.060)	**205.254** (0.326)

Table 20: Prediction MSE (mean and SE over 10 runs) of the estimated OT map for the OT map (d) $T_4(\mathbf{x}) = (4x_i^3)_{i\in[d]}$ based on $n = m = 5{,}000$ independent training sample points from the source and target distribution. Per HyCNN architecture $(44 \times L)$, the τ variant with the smallest Sinkhorn divergence validation metric ($\epsilon = 0.1$) for 1,000 validation samples at that dimension is used; see Table 21. Per dimension, the smallest mean MSE per method type is bold and colored, with the three smallest values across all methods underlined.

Method \ d		10	20	50
Entropic OT	$\epsilon = 0.1$	8.990 (0.046)	32.534 (0.144)	128.256 (0.323)
	$\epsilon = 1$	**3.417** (0.029)	**16.729** (0.092)	98.421 (0.422)
	$\epsilon = 10$	16.439 (0.073)	32.839 (0.138)	**82.155** (0.217)
Monge gap	(104×2)	10.503 (0.275)	**18.845** (0.241)	56.263 (0.193)
	(104×4)	**10.297** (0.227)	19.113 (0.190)	52.628 (0.244)
	(104×6)	12.489 (0.314)	20.964 (0.265)	**51.901** (0.165)
ICNN (ReLU)	(64×2)	5.675 (0.143)	14.824 (0.200)	68.621 (2.424)
	(64×4)	4.821 (0.081)	13.891 (0.320)	46.814 (0.934)
	(64×6)	**3.909** (0.080)	**11.780** (0.297)	**41.207** (0.559)
ICNN (L-ReLU)	(64×2)	6.677 (0.120)	21.465 (0.546)	4.5e+03 (2.3e+03)
	(64×4)	5.140 (0.131)	13.069 (0.265)	65.749 (1.899)
	(64×6)	**4.823** (0.163)	**12.307** (0.373)	**44.342** (0.820)
ICNNq (ReLU)	(64×2)	1.754 (0.042)	10.111 (0.121)	**25.761** (0.543)
	(64×4)	**1.435** (0.049)	**8.165** (0.616)	437.974 (50.407)
	(64×6)	1.746 (0.155)	22.471 (3.388)	1.82e+04 (2.4e+03)
ICNNq (Softplus)	$(64 \times 2); \tau = 0.1$	0.878 (0.038)	8.627 (0.049)	26.297 (0.826)
	$(64 \times 2); \tau = 1$	0.546 (0.064)	7.295 (0.422)	26.434 (0.609)
	$(64 \times 2); \tau = 10$	3.544 (0.021)	7.969 (0.035)	22.885 (0.060)
	$(64 \times 4); \tau = 0.1$	0.722 (0.012)	5.543 (0.209)	53.773 (3.493)
	$(64 \times 4); \tau = 1$	**0.360** (0.011)	3.532 (0.440)	36.023 (0.716)
	$(64 \times 4); \tau = 10$	3.348 (0.044)	7.731 (0.036)	22.116 (0.037)
	$(64 \times 6); \tau = 0.1$	0.812 (0.030)	5.330 (0.183)	2.96e+03 (1.12e+03)
	$(64 \times 6); \tau = 1$	0.412 (0.025)	**1.913** (0.187)	30.990 (0.838)
	$(64 \times 6); \tau = 10$	3.437 (0.037)	7.611 (0.027)	**21.854** (0.067)
HyCNN	(44×2)	2.757 (0.177)	7.841 (0.016)	20.554 (0.066)
	(44×4)	1.528 (0.171)	**7.783** (0.020)	**19.830** (0.028)
	(44×6)	**0.813** (0.029)	7.865 (0.028)	20.132 (0.044)

Table 21: Sinkhorn divergence (mean and SE over 10 runs) of the estimated OT map for the OT map (d) $T_4(\mathbf{x}) = (4x_i^3)_{i\in[d]}$. Per HyCNN architecture $(44 \times L)$ and per dimension, the τ variant with the smallest mean Sinkhorn divergence is bold and colored, with the three smallest values across all HyCNN rows underlined.

Method \ d		10	20	50
HyCNN	$(44 \times 2); \tau = 0.1$	**9.940** (0.121)	37.842 (0.130)	141.020 (0.355)
	$(44 \times 2); \tau = 1$	10.729 (0.029)	36.647 (0.087)	133.277 (0.744)
	$(44 \times 2); \tau = 10$	10.455 (0.033)	**35.179** (0.076)	**131.933** (0.176)
	$(44 \times 4); \tau = 0.1$	**9.574** (0.147)	37.305 (0.091)	144.998 (0.910)
	$(44 \times 4); \tau = 1$	10.544 (0.030)	35.759 (0.081)	135.727 (0.193)
	$(44 \times 4); \tau = 10$	10.438 (0.035)	**35.055** (0.083)	**131.051** (0.202)
	$(44 \times 6); \tau = 0.1$	**9.196** (0.037)	37.403 (0.086)	143.907 (0.621)
	$(44 \times 6); \tau = 1$	10.521 (0.034)	35.508 (0.082)	132.300 (0.195)
	$(44 \times 6); \tau = 10$	10.440 (0.035)	**35.100** (0.084)	**131.360** (0.199)

D.3.3 Ablations on convexity, architecture, initialization, and smoothness

In this subsection, we examine the effect of enforcing input-convexity, the choice of the architecture for the neural OT map, the initialization strategy, and the degree of smoothness in the activation functions on the performance of the neural OT map. To this end, we focus on the performance of different semi-dual estimators for the high-dimensional setting, $d = 50$, and the four ground truth OT potentials $\phi_1, \dots, \phi_4$ used in the earlier experiments.

Experimental setup. Concretely, we compare unconstrained MLP potentials (ReLU or Softplus, width 72), one-lane ICNNs (Softplus, width 64) with and without quadratic first-layer terms, and HyCNNs (log-sum-exp, width 44). The MLP here parameterizes a scalar potential whose gradient estimates the transport map (unlike the Monge gap MLP estimator above). We consider depths $L \in \{2, 4, 6\}$ and, for smooth activations, $\tau \in \{0.1, 1, 10\}$, with $\mathrm{Softplus}_\tau(a) = \tau \log(1 + e^{a/\tau})$. For HyCNNs, we compare our initialization from Section 2 with a near-ICNN initialization, i.e., where for each HyCNN block the hidden-to-hidden weights in the second-lane are initialized to 10^{-5} and the input-affine weights and biases are set to zero. This retains two trainable lanes while starting close to a one-lane network. The details to the training procedure, including learning rates, batch sizes, and optimization steps, follow those from the previously conducted high-dimensional experiments. The results of the respective numerical experiments are displayed in Table 22. All entries are test prediction MSEs, with coordinate errors summed, reported as mean (SE) over ten runs.

Main results. Overall, the HyCNN architecture with our initialization consistently achieves the best performance across the considered settings. Our ablation studies reveal that convexity helps stabilize the performance and smoothness aids in improving the overall accuracy, and that proper initialization is crucial for leveraging the HyCNN architecture most effectively. Further, according to the Sinkhorn-validated selection criterion, the best HyCNN configuration is consistently identified on an independent validation set (Table 23), and leads to the best overall performance across all four considered ground truth potentials.

On convexity and smoothness. The results reveal that without imposing convexity or smoothness, numerical instabilities occur. This may be a consequence of the MLP overfitting too much and its gradient being too unstable. Overall, imposing smoothness appears to consistently help performance. In contrast, imposing convexity rather stabilizes the performance across different settings and leads to performance gains for deeper architectures.

On the architecture. Standard ICNNs seem to be consistently outperformed by some smooth MLP, but do not experience any catastrophic loss of performance such as non-smooth MLPs. Further, deeper ICNNs perform best ($L = 6$) with moderate smoothness $\tau = 1$. We attribute this to the fact that standard ICNNs need to represent the respective quadratic nature of the ground truth potentials for which larger depth is beneficial and because the initialization by Hoedt & Klambauer (2023) is not optimized for large $\tau = 10$. However, the considered ICNNs are consistently outperformed by deeper HyCNNs with HyCNN initialization and under near-ICNN initialization with large-smoothness $\tau = 10$.

On the initialization. Comparing the two HyCNN-based estimators with near-ICNN and our proposed initialization, we see that choosing the latter leads to consistently better performance. Notably, except for depth $L = 6$ and $\tau = 10$, HyCNNs with near-ICNN initialization appear to be consistently outperformed by the corresponding ICNN architecture with same depth and smoothness parameter. This suggests that the architecture of HyCNNs is not the sole reason for their performance but also its initialization.

Table 22: Ablation studies for neural-potential OT estimators at $d = 50$, with $n = m = 5{,}000$. Columns correspond to the identity, anisotropic quadratic, quadratic plus ℓ_1, and quartic potentials, denoted by ϕ_1–ϕ_4. Entries display the mean (standard error) of the prediction MSE over 10 runs on 1,000 independent test points. Bold colored entries mark the smallest mean within each method and potential, and underlined values mark the three smallest means per column across each ground truth OT potential ϕ_i for $i = 1, \ldots, 4$.

Method	Width × depth; activation	ϕ_1	ϕ_2	ϕ_3	ϕ_4
MLP (ReLU)	(72×2); ReLU	**74.57** (3.04)	**72.98** (2.83)	**179.67** (6.71)	**169.48** (7.50)
	(72×4); ReLU	7.9×10^4 (4.0×10^4)	2.7×10^4 (3620.98)	3.6×10^5 (2.4×10^5)	1.4×10^7 (2.5×10^6)
	(72×6); ReLU	1.3×10^{10} (9.5×10^9)	1.1×10^{12} (1.1×10^{12})	2.0×10^{16} (1.4×10^{16})	3.8×10^{18} (1.7×10^{18})
MLP (Softplus)	(72×2); $\tau = 0.1$	52.50 (1.92)	49.73 (1.90)	149.94 (5.59)	116.86 (3.24)
	(72×2); $\tau = 1$	6.47 (0.30)	6.65 (0.17)	33.06 (0.33)	**22.34** (0.03)
	(72×2); $\tau = 10$	**0.70** (0.02)	**1.29** (0.07)	**20.16** (0.06)	25.42 (0.47)
	(72×4); $\tau = 0.1$	3.3×10^4 (2.0×10^4)	1.9×10^4 (6028.90)	1.1×10^5 (4.5×10^4)	1.0×10^6 (2.6×10^5)
	(72×4); $\tau = 1$	10.28 (1.01)	11.56 (1.87)	43.07 (0.96)	24.19 (0.18)
	(72×4); $\tau = 10$	0.93 (0.14)	5.4×10^8 (5.4×10^8)	21.76 (0.08)	25.26 (0.39)
	(72×6); $\tau = 0.1$	3.1×10^9 (3.1×10^9)	4.6×10^6 (3.1×10^6)	1.9×10^{11} (1.5×10^{11})	7.0×10^{12} (3.8×10^{12})
	(72×6); $\tau = 1$	21.94 (5.83)	19.14 (4.31)	41.96 (2.04)	24.43 (0.46)
	(72×6); $\tau = 10$	1.03 (0.12)	1.67 (0.10)	22.72 (0.32)	26.64 (0.47)
ICNN (Softplus)	(64×2); $\tau = 0.1$	18.56 (0.45)	18.92 (0.41)	102.28 (2.20)	63.68 (1.26)
	(64×2); $\tau = 1$	6.73 (0.07)	6.29 (0.07)	42.26 (0.87)	27.12 (0.21)
	(64×2); $\tau = 10$	54.62 (8.41)	36.67 (3.19)	116.14 (24.68)	153.74 (26.45)
	(64×4); $\tau = 0.1$	10.60 (0.25)	10.31 (0.23)	59.62 (0.97)	44.71 (0.61)
	(64×4); $\tau = 1$	3.24 (0.09)	3.12 (0.08)	30.95 (0.38)	23.90 (0.13)
	(64×4); $\tau = 10$	11.94 (1.17)	5.64 (0.36)	36.76 (0.99)	67.77 (4.23)
	(64×6); $\tau = 0.1$	7.77 (0.17)	7.48 (0.11)	47.15 (0.71)	38.24 (0.82)
	(64×6); $\tau = 1$	**2.11** (0.04)	**2.06** (0.03)	**26.09** (0.13)	**21.97** (0.10)
	(64×6); $\tau = 10$	13.86 (1.82)	4.57 (0.26)	32.82 (1.26)	48.72 (2.13)
HyCNN (near-ICNN init)	(44×2); $\tau = 0.1$	19.98 (0.37)	20.54 (0.40)	104.17 (2.64)	66.95 (1.35)
	(44×2); $\tau = 1$	10.55 (0.21)	10.07 (0.10)	61.62 (0.75)	36.45 (0.60)
	(44×2); $\tau = 10$	5.02 (0.71)	18.49 (3.53)	51.19 (7.17)	68.15 (10.99)
	(44×4); $\tau = 0.1$	12.27 (0.18)	12.25 (0.23)	64.82 (0.66)	50.35 (1.00)
	(44×4); $\tau = 1$	4.86 (0.10)	4.83 (0.08)	36.25 (0.38)	25.95 (0.17)
	(44×4); $\tau = 10$	0.83 (0.04)	0.98 (0.07)	20.69 (0.08)	27.54 (0.55)
	(44×6); $\tau = 0.1$	9.27 (0.20)	9.26 (0.19)	53.56 (1.62)	44.98 (3.18)
	(44×6); $\tau = 1$	3.60 (0.08)	3.56 (0.07)	31.09 (0.31)	24.09 (0.25)
	(44×6); $\tau = 10$	**0.76** (0.03)	**0.85** (0.04)	**20.53** (0.04)	**23.87** (0.27)
HyCNN (HyCNN init)	(44×2); $\tau = 0.1$	18.31 (0.24)	18.30 (0.22)	84.77 (0.53)	58.38 (0.44)
	(44×2); $\tau = 1$	8.60 (0.06)	8.85 (0.06)	50.00 (0.30)	31.06 (0.16)
	(44×2); $\tau = 10$	0.78 (0.01)	0.71 (0.01)	21.93 (0.08)	20.52 (0.05)
	(44×4); $\tau = 0.1$	10.62 (0.07)	10.77 (0.09)	56.84 (0.40)	43.54 (0.35)
	(44×4); $\tau = 1$	3.96 (0.02)	4.07 (0.05)	31.68 (0.13)	24.38 (0.06)
	(44×4); $\tau = 10$	0.40 (0.006)	**0.42** (0.01)	19.89 (0.04)	**19.87** (0.04)
	(44×6); $\tau = 0.1$	7.17 (0.09)	7.31 (0.09)	44.48 (0.47)	36.12 (0.25)
	(44×6); $\tau = 1$	2.94 (0.07)	2.98 (0.07)	27.75 (0.23)	22.62 (0.14)
	(44×6); $\tau = 10$	**0.38** (0.005)	0.51 (0.01)	**19.80** (0.02)	20.11 (0.05)

Table 23: Prediction MSE (mean and standard error across 10 runs) based on Sinkhorn-validation based selection criterion among depth and smoothness for each method and potential from Table 22. Each cell shows the mean and standard error of the selected method. Bold text denotes the smallest selected mean in each column.

Method	ϕ_1	ϕ_2	ϕ_3	ϕ_4
MLP (ReLU)	74.57 (3.04) $L = 2$; ReLU	72.98 (2.83) $L = 2$; ReLU	179.67 (6.71) $L = 2$; ReLU	169.48 (7.50) $L = 2$; ReLU
MLP (Softplus)	0.70 (0.02) $L = 2$; $\tau = 10$	1.29 (0.07) $L = 2$; $\tau = 10$	20.16 (0.06) $L = 2$; $\tau = 10$	22.34 (0.03) $L = 2$; $\tau = 1$
ICNN (Softplus) (no quadratic)	2.11 (0.04) $L = 6$; $\tau = 1$	2.06 (0.03) $L = 6$; $\tau = 1$	42.26 (0.87) $L = 2$; $\tau = 1$	27.12 (0.21) $L = 2$; $\tau = 1$
HyCNN (near-ICNN init)	0.83 (0.04) $L = 4$; $\tau = 10$	0.98 (0.07) $L = 4$; $\tau = 10$	20.69 (0.08) $L = 4$; $\tau = 10$	36.45 (0.60) $L = 2$; $\tau = 1$
HyCNN (HyCNN init)	**0.38** (0.005) $L = 6$; $\tau = 10$	**0.42** (0.01) $L = 4$; $\tau = 10$	**19.80** (0.02) $L = 6$; $\tau = 10$	**19.87** (0.04) $L = 4$; $\tau = 10$

E Single Cell Data Analysis

A central task in modern functional genomics and drug discovery is to predict how a population of cells responds to an external perturbation such as a chemical compound, a genetic intervention, or a change in culture conditions. Because most profiling technologies (single-cell RNA sequencing, mass cytometry, multiplexed imaging) destroy the cell upon measurement, one cannot observe the same cell before and after treatment. As a consequence, the unperturbed *control* population and the *perturbed* population are only accessible through two independent collections of single-cell measurements, and the learning problem is intrinsically one of matching *distributions* rather than individual pairs of data points. This viewpoint was advocated by Schiebinger et al. (2019), who used optimal transport to infer developmental trajectories from unpaired snapshots of single-cell gene expression during reprogramming, and was subsequently taken up in a series of works on perturbation response modeling, most notably Bunne et al. (2023). These works established optimal transport as a natural and principled language for formulating single-cell response prediction.

The 4i dataset. The dataset we consider is the multiplexed imaging dataset of Bunne et al. (2023), obtained from the *iterative indirect immunofluorescence imaging* (4i) technology. It is based on the sequential application of indirect immunofluorescence staining cycles to the same specimen, which allows a very large number of protein markers to be probed on the same set of cells. The result is a high-dimensional, single-cell-resolved readout of the proteomic state of each cell under control and under a wide range of drug treatments, made available by Bunne et al. (2023) in a preprocessed form that we use without further modification. For each drug treatment we obtain two point clouds in a common feature space, one from the untreated control population and one from the cells that were exposed to the drug, with sample sizes between roughly 1,300 and 3,000 cells per treatment (see Table 24).

Mathematical task. Denote by P the distribution of the control cell population and by Q_D the distribution of the cell population after exposure to drug D. We are given measurements $\mathbf{X}_1, \dots, \mathbf{X}_n$ from the control cell population and after treatment D, denoted by $\mathbf{Y}_1, \dots, \mathbf{Y}_m$, which we model as independent and identically distributed realizations from P and Q_D, respectively. We wish to construct an estimator $\widehat{T}: \mathbb{R}^p \to \mathbb{R}^p$ such that the pushforward $\widehat{T}_{\#}P$ is as close as possible to Q_D. The estimator is evaluated on a held-out test partition consisting of samples from both P and Q_D, and the quality of the fit is measured by the Sinkhorn divergence, a geometrically meaningful two-sample discrepancy measure between probability distributions (Feydy et al., 2019; Peyré & Cuturi, 2019), between $\widehat{T}_{\#}\widehat{P}_{\text{test}}$ and $\widehat{Q}_{D,\text{test}}$.

Why optimal transport. Among distribution matching methodologies, optimal transport is particularly appealing in this setting for two reasons. Conceptually, the static OT map minimizes the expected squared displacement and thus embodies the biological belief that cells evolve and proliferate along minimal-effort trajectories in feature space, rather than being reshuffled arbitrarily. This heuristic is

motivated by the presence of an underlying potential, which drives the development and change of cells as they undergo the effects of a treatment. Methodologically, OT has been repeatedly reported to perform well as a predictive framework for single-cell perturbation response, starting from the developmental-trajectory analysis of Schiebinger et al. (2019) and including the neural OT approach of Bunne et al. (2023), among others. We therefore adopt neural OT map estimation as the basis of our analysis and compare our HyCNN architecture against the most competitive convex neural OT baseline identified in our synthetic experiments.

Concrete modeling choices. For each drug D, we train two families of neural OT estimators to map the control distribution onto the perturbed distribution: a HyCNN of width 44 and depth $L = 4$, and an ICNN of width 64 and depth $L = 4$ equipped with softplus activations and a quadratic term in the first layer (ICNN(sp+q)). The ICNN(sp+q) variant consistently emerged as one of the strongest convex neural baselines in our synthetic OT experiments, see Section D.3, which motivates its use as the reference competitor here. The width–depth configuration 64×4 was chosen because depth four has become a de facto standard in the neural OT literature (Makkuva et al., 2020; Bunne et al., 2023; Uscidda & Cuturi, 2023), and because the corresponding width of HyCNNs yields models whose parameter size is comparable to the ICNN. For the HyCNN, we employ a smooth log-sum-exp activation with a smoothing parameter $\tau = 1$, a choice that is capable of adapting to smooth structure while remaining flexible. The ICNN baseline uses softplus activations with $\tau = 1$ for the same reason.

Learning algorithm. For training, we first split the control and drug cell data into $70\% - 10\% - 20\%$ for training, validation, and testing, respectively. We estimate the OT map using the variational semidual formulation as in (6) for HyCNN and (45) for ICNN, in which the outer and inner potentials are both parameterized by a HyCNN or ICNN. For the HyCNN and ICNN, we impose essentially the same learning algorithm as in Section D.3. To assess the effect of the training budget, we consider two complementary schedules that are applied identically to both architectures. The *short* schedule trains the networks for $S_{\text{out}} = 1{,}000$ outer iterations with $S_{\text{in}} = 5$ inner iterations per outer step, using the Adam optimizer with $\beta_1 = 0.5$, $\beta_2 = 0.9$, initial learning rate $\lambda = 10^{-2}$, and a cosine schedule with final learning-rate ratio 10^{-2}. The *long* schedule trains the networks for $S_{\text{out}} = 50{,}000$ outer iterations with $S_{\text{in}} = 10$ inner iterations per outer step, using the Adam optimizer with the same momentum parameters and a constant learning rate $\lambda = 5 \times 10^{-4}$. The long schedule is roughly 100 times more expensive than the short one (per-treatment training times of approximately 40s, 60s, 2h, and 3h for the short-trained ICNN, short-trained HyCNN, long-trained ICNN, and long-trained HyCNN, respectively, on our hardware); for this reason the long schedule is run for a single random seed per treatment, whereas the short schedule is repeated for 50 seeds. All remaining hyperparameters are kept identical across the two architectures to keep the comparison as controlled as possible.

Evaluation and model selection. To monitor and select models during training, we evaluate each candidate on a dedicated validation partition drawn from the training set. The validation criterion is the Sinkhorn divergence $\mathcal{S}\epsilon$ with squared Euclidean cost and regularization $\epsilon = 0.1$ (Feydy et al., 2019; Peyré & Cuturi, 2019) between the predicted pushforward $\widehat{T}_{\#}\widehat{P}_{\text{val}}$ and the ground-truth validation target $\widehat{Q}_{D,\text{val}}$. Validation losses are recorded every 10 outer iterations under the short schedule, yielding 100 candidate checkpoints per run, and every 100 outer iterations under the long schedule, yielding 500 candidate checkpoints per run. For each run, we identify the $K = 10$ checkpoints with the smallest validation Sinkhorn divergence and report the mean test Sinkhorn divergence over the 20% held-out test partition. Averaging over the ten best-validation checkpoints, rather than selecting the single best one, ensures that the reported value reflects an entire neighborhood of the validation optimum and confirms that the comparison is stable along the training trajectory rather than an artifact of a single lucky checkpoint. This *best-validation averaging* rule is applied identically to both architectures. Under the short schedule, we repeat each experiment over 50 random seeds governing data splits and network initialization, and report cross-seed means together with standard errors. Under the long schedule, the same rule applies, but due to the substantially higher per-run computational cost, we conduct only a single run per treatment, so no standard error is reported.

Results and discussion. The resulting test-set Sinkhorn divergences are collected in Table 24, which reports both training schedules for each of the 35 drug treatments. Each table entry is the test-set Sinkhorn divergence averaged over the $K = 10$ checkpoints with the smallest validation loss along the corresponding training run, and this best-validation averaging is applied identically to both architectures and both schedules. The left two columns show, for each treatment, the cross-seed mean of this checkpoint-averaged quantity (with standard error in parentheses) over 50 random seeds under the short schedule. The right two columns show the same checkpoint-averaged quantity for the single

random-seed run under the long schedule, for which no standard error is reported. The results of the first seed of the short schedule, together with the long-schedule run, are visualized as grouped bar charts in Figure 5**(b)**; see Figure 17 for an enlarged version. In the figures, treatments are ordered by short-trained HyCNN test performance.

Three patterns emerge consistently from Table 24. First, HyCNN attains a smaller test-set Sinkhorn divergence than ICNN consistently across treatments and schedules. Under the short schedule, HyCNN strictly improves over ICNN on all 35 treatments, with (cross-seed) mean improvements that are typically within the range 25-40% and outside one standard error. Under the long schedule, HyCNN remains the strictly better architecture on 31 of the 35 treatments, with the four exceptions (Dasatinib, Imatinib, Pomalidomide-carfilzomib-dexamethasone, and Regorafenib) showing only marginal differences in favor of ICNN. Since each entry already averages over the ten best validation checkpoints, these observations reflect a consistent improvement between the two architectures along their respective training trajectories. Second, the long schedule lowers the test-set Sinkhorn divergence relative to the short schedule across nearly every treatment for both architectures, confirming that the additional optimization budget is genuinely informative rather than an artifact of the learning-rate schedule. Third, the short-trained HyCNN performs competitively with the long ICNN despite using roughly 100 times less compute. It matches or improves upon the long ICNN on 14 of the 35 treatments and is often outperformed only by small margins on the remaining ones. Altogether, these findings indicate that, within the neural OT framework, replacing the ICNN potential by a HyCNN consistently improves the distributional fit, and that this improvement is not an artifact of an under-trained ICNN baseline. This is a useful insight for practitioners of neural OT in single-cell applications, as the resulting gain comes at only a slight additional per-iteration cost due to the two-gate architecture of HyCNN blocks.

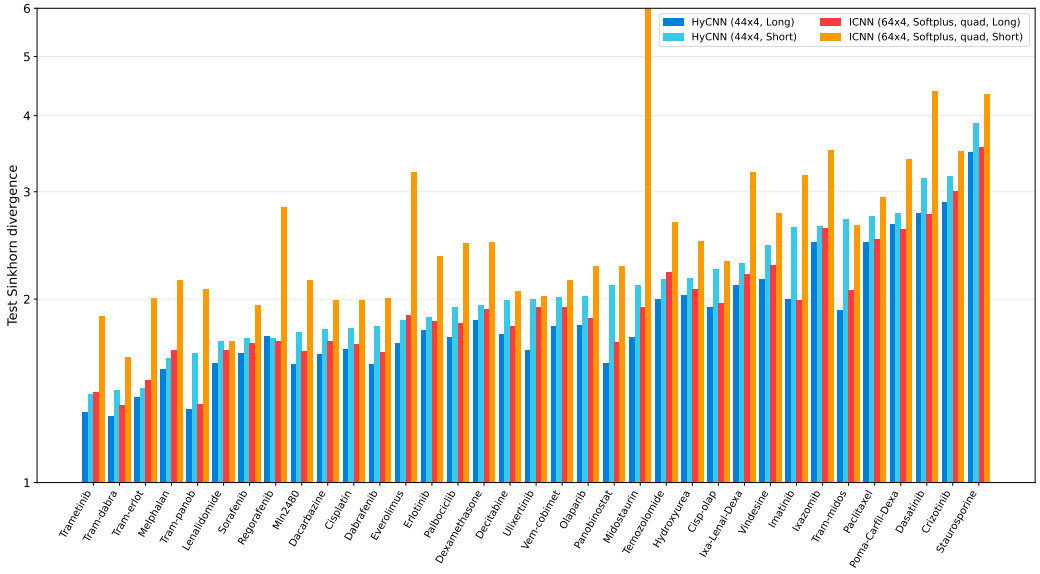

Figure 17: Enlarged view of Figure 5**(b)** with drug labels on the x-axis. Sinkhorn divergence between predicted and ground-truth drug-perturbed distributions on a 20% held-out test partition of the 4i dataset for each drug, based on ICNN (64×4, Softplus, $\tau = 1$, quad) and HyCNN (44×4, log-sum-exp, $\tau = 1$), trained for $S_{\text{out}} = 1{,}000$ outer iterations with $S_{\text{in}} = 5$ inner iterations (short) and $S_{\text{out}} = 50{,}000$ outer iterations with $S_{\text{in}} = 10$ inner iterations (long). For each architecture and schedule, the displayed value is the test-set Sinkhorn divergence averaged over the $K = 10$ checkpoints with smallest validation loss along the corresponding training run. Per treatment, the distributional fit is computed on a fixed train/validation/test split, for which reason the values for the short runs slightly deviate from the cross-seed means reported in Table 24. We use the following abbreviations: Cisp-olap for Cisplatin-olaparib, Ixa-Lenal-Dexa for Ixazomib-lenalidomide-dexamethasone, Poma-Carfil-Dexa for Pomalidomide-carfilzomib-dexamethasone, Tram-dabra for Trametinib-dabrafenib, Tram-erlot for Trametinib-erlotinib, Tram-midos for Trametinib-midostaurin, Tram-panob for Trametinib-panobinostat, and Vem-cobimet for Vemurafenib-cobimetinib.

Table 24: Sample sizes and test-set prediction errors across drug treatments for ICNN (64×4) with softplus ($\tau = 1$) and a quadratic term in the first layer, and HyCNN (44×4) with log-sum-exp ($\tau = 1$). The two architectures are compared under two training budgets: the short run ($S_{\text{out}} = 1{,}000$ outer iterations, $S_{\text{in}} = 5$ inner steps, initial learning rate $\lambda = 0.01$, final learning-rate ratio 0.01 under a cosine scheduler), and the long run ($S_{\text{out}} = 50{,}000$ outer iterations, $S_{\text{in}} = 10$ inner steps, constant learning rate $\lambda = 5 \times 10^{-4}$). For each drug, we report the test-set Sinkhorn divergence ($\epsilon = 0.1$) on a 20% test partition, averaged over the $K = 10$ checkpoints with smallest loss on a 10% validation partition; validation losses are recorded every 10 outer iterations under the short schedule and every 100 outer iterations under the long schedule. The left two columns report the cross-seed mean of this checkpoint-averaged quantity (with standard error in parentheses) across 50 random seeds; the right two columns report the same checkpoint-averaged quantity for the single long-schedule run per treatment. Bold indicates the smaller mean within each $(S_{\text{out}}, S_{\text{in}})$ pair. Treatments are abbreviated as in Figure 17.

Treatment	Sample size	Sinkhorn Divergence ($\epsilon = 0.1$)			
		ICNN(sp+q) ($S_{\text{out}} = 1{,}000,\ S_{\text{in}} = 5$)	HyCNN ($S_{\text{out}} = 1{,}000,\ S_{\text{in}} = 5$)	ICNN(sp+q) ($S_{\text{out}} = 50{,}000,\ S_{\text{in}} = 10$)	HyCNN ($S_{\text{out}} = 50{,}000,\ S_{\text{in}} = 10$)
Cisplatin	2693	2.86 (0.63)	**1.75 (0.011)**	1.69	**1.66**
Cisp-olap	2245	2.65 (0.13)	**2.04 (0.019)**	1.97	**1.94**
Crizotinib	1978	5.35 (0.97)	**3.19 (0.041)**	3.00	**2.88**
Dabrafenib	2675	2.57 (0.41)	**1.64 (0.012)**	1.64	**1.57**
Dacarbazine	2485	2.49 (0.25)	**1.67 (0.013)**	1.71	**1.62**
Dasatinib	2250	5.13 (0.55)	**3.19 (0.038)**	**2.75**	2.76
Decitabine	2648	2.54 (0.16)	**1.92 (0.014)**	1.80	**1.75**
Dexamethasone	2796	2.88 (0.29)	**1.95 (0.018)**	1.93	**1.84**
Erlotinib	2406	3.32 (0.47)	**1.91 (0.014)**	1.84	**1.78**
Everolimus	2592	3.23 (0.6)	**1.85 (0.011)**	1.88	**1.69**
Hydroxyurea	2715	2.95 (0.35)	**1.98 (0.012)**	2.08	**2.03**
Imatinib	2631	3.24 (0.3)	**2.32 (0.023)**	**2.00**	**2.00**
Ixazomib	2514	3.72 (0.18)	**2.79 (0.019)**	2.62	**2.48**
Ixa-Lenal-Dexa	2522	3.40 (0.32)	**2.16 (0.013)**	2.20	**2.10**
Lenalidomide	2524	2.36 (0.26)	**1.56 (0.011)**	1.65	**1.57**
Melphalan	2352	3.75 (1.3)	**1.71 (0.012)**	1.65	**1.53**
Midostaurin	2425	3.24 (0.26)	**2.07 (0.018)**	1.94	**1.73**
Mln2480	2367	2.47 (0.3)	**1.55 (0.014)**	1.64	**1.56**
Olaparib	2540	3.50 (0.54)	**2.01 (0.015)**	1.86	**1.81**
Paclitaxel	2636	4.72 (1.2)	**2.76 (0.019)**	2.51	**2.48**
Palbociclib	2322	3.01 (0.4)	**1.89 (0.015)**	1.83	**1.73**
Panobinostat	2492	2.95 (0.34)	**1.90 (0.018)**	1.70	**1.57**
Poma-Carfil-Dexa	2545	3.82 (0.23)	**2.70 (0.021)**	**2.61**	2.65
Regorafenib	2524	4.91 (1.7)	**1.75 (0.013)**	**1.71**	1.74
Sorafenib	2763	3.02 (0.53)	**1.66 (0.013)**	1.69	**1.63**
Staurosporine	1310	13.94 (8.8)	**4.08 (0.047)**	3.55	**3.48**
Temozolomide	2513	7.90 (5.5)	**1.92 (0.02)**	2.21	**2.00**
Trametinib	2637	3.42 (1.1)	**1.37 (0.0092)**	1.41	**1.31**
Tram-dabra	2958	2.23 (0.3)	**1.31 (0.011)**	1.34	**1.29**
Tram-erlot	2452	2.07 (0.14)	**1.39 (0.01)**	1.47	**1.38**
Tram-midos	2211	5.60 (1.3)	**2.59 (0.026)**	2.07	**1.92**
Tram-panob	2377	2.52 (0.37)	**1.56 (0.016)**	1.34	**1.32**
Ulixertinib	2858	2.27 (0.14)	**1.78 (0.014)**	1.94	**1.65**
Vem-cobimet	2695	2.70 (0.17)	**1.93 (0.018)**	1.94	**1.81**
Vindesine	2102	3.49 (0.41)	**2.65 (0.026)**	2.27	**2.16**

CellOT and future directions. For context, the CellOT pipeline of Bunne et al. (2023) trains the ICNN-based neural OT model for substantially more iterations than the $50{,}000$ outer steps used in our long schedule; for instance, Bunne et al. (2023) use $S_{\text{out}} = 250{,}000$ iterations with $S_{\text{in}} = 10$ inner loops at a constant learning rate of 10^{-4}. Our long schedule already addresses much of the concern that the ICNN baseline could simply be under-trained: increasing the ICNN budget from $1{,}000$ to $50{,}000$ outer iterations narrows its absolute error noticeably, yet HyCNN remains the better architecture on 31 of the 35 treatments under the long schedule, and the short-trained HyCNN remains broadly competitive with the long-trained ICNN despite using 100 times less compute. Since both architectures continue to benefit from additional training while preserving this ordering, we expect a full $250{,}000$-iteration evaluation against the CellOT baseline to reinforce rather than overturn our qualitative picture reported here. A comprehensive comparison with a larger budget is, therefore, a natural next step.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction (Section 1) clearly state the main contributions: (i) the HyCNN architecture and its convexity guarantee (Proposition 2.1), (ii) quantitative approximation advantages over ICNNs (Theorems 3.2 and 3.3), and (iii) empirical improvements in convex regression and OT map estimation (Section 5). The claims accurately reflect the scope of the theoretical and experimental results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated section is provided in Section 6, discussing: the inductive bias of the HyCNN architecture towards smooth coercive convex functions, and the limitations in terms of our computational capabilities to train networks for a large number of iterations. Training the networks for longer might lead to a smaller performance gap between other input convex networks and HyCNNs.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All theoretical results (Propositions 2.1 and 3.1, Theorems 3.2, 3.3, and 3.5) are formally stated with complete assumptions. Proposition 2.1 is proved in the main text. Full proofs of all remaining results are provided in Appendix B, with supporting technical lemmas.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: All experiments with synthetic data have fully specified generating processes (distributions, sample sizes, noise levels). The network architectures (widths, depths, activation functions, representations of weights), training procedures (optimizer, learning rate, batch size, epochs, iterations), and initialization schemes are described in detail in Section 5 and Appendices D, E. The algorithms for empirical risk minimization and for OT map learning are detailed in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We refer to the GitHub repository https://www.github.com/hundrieser/HyCNN where the code for reproducing relevant experiments is provided. All experiments with synthetic data have fully specified generating processes, and the architectural and training details are described in sufficient detail to allow reimplementation.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All training details are specified: Adam optimizer with explicit learning rate, β_1, β_2, batch sizes, number of epochs/iterations, initialization schemes for each architecture, learning rate schedules, and evaluation metrics. These are provided in Section 5 and in full detail in Appendices D, and E. This encompasses the details to understand the synthetic data experiments and the data analysis for the single-cell multiplexed imaging application on the 4i-dataset.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All main experiments are repeated 10 times with different random seeds for both data generation and network initialization. We report 10th–90th percentile bands in all figures (Figures 1, 4, 5 and appendix figures). The variability captured by these bands reflects both the randomness in the training data and the network initialization. Performance tables in Appendix D additionally report mean and standard error across the 10 runs, which is defined as the standard deviation divided by the square root of the number of runs. For the single-cell multiplexed imaging application in Appendix E, we increase the number of random seeds to 50 under the short training schedule and report the corresponding mean and standard error. To assess sensitivity to the training budget, we additionally evaluate a substantially longer schedule (T = 50,000 outer iterations, S = 10 inner iterations); the test-set Sinkhorn divergence obtained from the long-schedule run is consistent with the 50-seed mean of the short schedule for both ICNN and HyCNN. Moreover, to further ensure stability of validation checkpoints, we consider the average across the 10 best checkpoints (instead of the single best checkpoint). Due to the substantial computational cost of the long schedule, only a single random seed per treatment is reported for the long-schedule results. We acknowledge that the absence of multiple seeds for the long schedule is a limitation imposed by our current computational resources rather than by the methodology itself.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: At the beginning of Appendix D we specify the hardware used for all experiments. We always state the numbers of iterations. Beyond the experiments reported in the paper, the full research project required additional compute for preliminary and exploratory work that is not included in the final manuscript, including the development and validation of the HyCNN initialization scheme, hyperparameter exploration (e.g., learning rate, batch

size, number of epochs, width, and depth), ablations on skip-connection variants, as well as failed runs and debugging. All such runs were executed on the same hardware described above. We estimate that the total compute consumed during the project to reproduce the reported experiments exceeds the compute for the experiments reported in the paper by approximately a factor of three to five.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics. The work is foundational in nature, involving a novel neural network architecture and synthetic experiments, and does not involve human subjects, sensitive data, or dual-use concerns.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work introduces a foundational neural network architecture for learning convex functions. It is a methodological contribution with no direct negative societal implications. The applications considered (convex regression and optimal transport) are standard problems in computational mathematics and machine learning. Our application on the single-cell multiplexed imaging data is focused on improving the estimation of OT maps for better understanding cellular dynamics, which has potential positive impacts in biology and medicine. The risks of misuse of this work are minimal, as the technology is not directly applicable to harmful applications.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out

that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release pre-trained models, datasets, or any artifacts that pose a risk for misuse. The contribution is a neural network architecture evaluated on synthetic and real data.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper does not use pre-trained models. The 4i dataset is cited and we refer to the original paper for the license and terms of use. Except for the 4i dataset, all experiments are based on synthetically generated data. Prior architectures used for comparison (ICNNs, GroupMax networks, Monge Gap, entropic optimal transport map) are properly cited, and we refer to the `geomloss` and `ott-jax` libraries for the code used to implement these baselines.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new datasets or pre-trained models. The primary contribution is a neural network architecture, which is fully documented in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.

- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: ChatGPT Codex 5.4 and 6.1 Sol and Claude Code Opus 4.6 were used to assist in implementing the simulation code and generating figures. The generated code was carefully reviewed by a human and checked for correctness. LLMs were not used as a component of the core methodology (i.e., the HyCNN architecture, theoretical analysis, or experimental design).

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.