Positive Operator Circuits: a Quantum Information Theoretic Approach to Tractable Probabilistic Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

By recursively nesting sums and products, probabilistic circuits (PCs) have emerged in recent years as an attractive class of generative models as they enjoy, for instance, polytime marginalization of random variables. However, as these PCs form monotone functions, specifically the parameters are constrained to positive real values, their expressive power is limited. Drawing inspiration from the PC literature and quantum information theory we introduce a novel class of probabilistic models, which we dub *positive operator circuits* (POXs). Contrary to the parametrization of PCs (constrained to positive real values), POXs use complex valued parameters. By leveraging concepts from quantum information theory, we show that POXs encode proper probability distributions, while retaining the tractable marginalization property of PCs. We also give a prescription for constructing an efficient parametrization of POXs that enables us to perform parameter learning on hardware accelerators using standard gradient-based optimization. Lastly, we provide an implementation and experimental evaluation for POXs, which establishes POXs as an attractive class of tractable probabilistic models.

1 Introduction

Probabilistic circuits (PCs) [**?** **?**] belong to an unusual class of probabilistic models: they are highly expressive but at the same time also tractable. For instance, so-called decomposable probabilistic circuits [**?**] allow for the computation of marginals in time polynomial in the size of the circuit. From an abstract algebra perspective, PCs constitute computation graphs that perform their computations within the probability semiring. That is, they sum and multiply elements from the $[0, 1]$ interval with each other.

add back this shit at the end for replication and all that super long annoying list of silly questions

?] noted that it is exactly this restriction to positive values that limits the expressive efficiency (or succinctness) of PCs [**?** **?**]. In particular, the positivity constraint on the set of elements that PCs operate on prevents them from modelling negative correlations between variables. Circuits that are incapable of modelling negative correlations, i.e. circuits that can only combine probabilities in an additive fashion, are also called monotone circuits [**?**]. This restricted expressiveness can be combatted by the use of so-called *non-monotone* circuits, where subtractions are allowed as a third operation (besides sums and products). Interestingly, **?**] showed that a mere single subtraction can render non-monotone circuits exponentially more expressive than monotone circuits.

As shown by **?**], non-monotone circuits do, however, introduce an important complication: if non-monotone circuits are not designed carefully, verifying whether a circuit encodes a valid probability distribution or not is an NP-hard problem. This does also render learning the parameters of a circuit practically infeasible.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

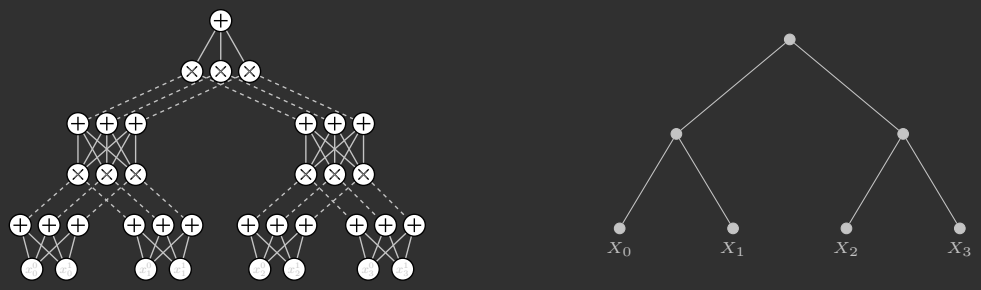

Figure 1: Left: Layered PC over four binary variables X_i with $i \in \{0, 1, 2, 3\}$ taking values x_i^0 or x_i^1. Within each partition in the bottom layer three mixtures for each of the four X_i's are constructed using weighted sums (weights are not shown in the graphical representation but are present on the edges feeding into sum units). The sum and product units in the circuit are the elemental constitute the elemental computing units and make the partitions. The circuit in Figure 1 has, except at the very top and bottom, three components in each partition. As increasing the number of components per partition increases the number of parameters (weights on edges feeding into sum units), the number of components per partition controls the capacity of a circuit. Right: partition tree abstracting the layered PC on the right.

We will formulate, in a first instance, the class of layered probabilistic circuits (cf. **? ?**]) in terms of, what we call, partition circuits (Section 2). This will allow us, Together with a slight generalization of concepts from quantum information theory (Section 3), to concisely introduce *positive operator circuits* (POXs) – a novel non-monotone circuit class that uses complex valued parameters (instead of positive real ones) and that forms by construction valid probability distributions (Section 4). Lastly, we demonstrate the benefit of modelling probability distributions using positive operator circuits (Section 6), in which POXs outperform competing (tractable) density estimators on datasets from the MNIST family but where we also point out important open questions. In Section 5 we study the related work, and we give concluding remarks in Section 7.

2 Partition Circuits

Inspired by simple feed-forward neural networks, **? ?**] introduced the concept of layered probabilistic circuits. These layered circuits are amenable to trivial parallelization and can be run on modern discrete GPUs. In Figure 1 we give a graphical representation of such a layered circuit where we follow the construction introduced by **?**]. For the sake of conciseness we describe here only so-called *structured decomposable and smooth* PCs. For a broader overview we refer the reader to the excellent work of **?**].

Layers within a layered PC constitute blocks of computational units that are processed sequentially in a bottom-up fashion. Layers consist themselves of so-called partitions. The circuit in Figure 1 has four partitions in the leaf layer (very bottom), two in the subsequent sum and product layers, and a single partition in the last product layer and in the final root node (top most sum). By construction, partitions in the same layer have disjoint scopes. That is, they are functions over disjoint sets of variables. This property is called structured decomposability in the PC literature [**?**]. Note also that layered PCs are by construction smooth [**?**]: the union of the scopes of the partitions within a layer is exhaustive. That is, the union of the scopes equals the set of variables given as input to the circuit.

A (layered) PC is then parametrized by weighing the inputs to the sum units with positive real numbers – giving rise to an unnormalized probability distribution over the input variables. Due to the properties of (structured) decomposability and smoothness the distribution can be normalized in time polynomial in the size of the circuit [**?**]. Thanks to the properties of smoothness and (structured) decomposability we can also marginalize out single variables from a PC. Again in time polynomial in the size of the circuit.

Recently, **?**] introduced the concept of *partition trees* to abstract away certain aspects of smooth and structured decomposable probabilistic circuits. We give such a partition tree on the right side of Figure 1. We now take this abstraction a step further and define the computations performed by the probabilistic circuit not on the atomic computation units of the circuit itself (i.e. sum and

product units) but use the nodes in the partition tree as the elemental computation units. In other words, we regard the partition tree as a computation graph. To make this distinction explicit we dub these computation graphs *partition circuits*. For the sake of conciseness, we will limit ourselves in this paper to balanced binary partition trees and will assume that the number of input variables is a power of two (unless indicated otherwise). Consequently, we will study partition circuits of depth $\log_2 M + 1$ where M is the number of input variables.

Definition 2.1 (Partition Circuit)**.** *A partition circuit over a set of variables is a parametrized computation graph taking the form of a binary tree. The partition circuit consists of three kinds of computation units: leaf and internal units, as well as a single root. Units at the same distance from the root form a layer. Furthermore, let p_n denote the root unit or an internal unit. The unit p_n then receives its inputs from two units in the previous layer, which we denote by p_{n_l} and p_{n_r}. Each computation unit is input to exactly one other unit, except the root unit, which is the input to no other unit.*

For the interested reader we give the definition of a layered probabilistic circuit in terms of a partition circuit, as well as a proof that they form valid probability distributions, in Appendix 2.

3 Adapting Quantum Information Theory for Probabilistic Modelling

A widely used and elegant framework to describe measurements of quantum systems is the so-called *positive operator-valued measure* (POVM) formalism. While POVMs have physical interpretations in terms of quantum information and quantum statistics, we will only be interested in their mathematical properties as we will use them to show that positive operator circuits (defined in Section 4) form valid probability distributions. We refer the reader to [**?**] for an in-depth exposition on the topic, as well as quantum computing and quantum information theory in general.

Definition 3.1 (Positive Semidefinite)**.** *An $N \times N$ Hermitian matrix H is called positive semidefinite (PSD) if and only if $\forall \mathbf{x} \in \mathbb{C}^N : \mathbf{x}^* H \mathbf{x} \geq 0$, where $\mathbf{x}^*$ denotes the conjugate transpose and $\mathbb{C}^N$ the N-dimensional space of complex numbers.*

Definition 3.2 (POVM [**?** , Page 90])**.** *A positive operator-valued measure is a set of PSD matrices $\{E(i)\}_{i=0}^{V-1}$ ($V \in \mathbb{N}$ being the number of possible measurement outcomes) that sum to the identity:*

$$\sum_{i=0}^{V-1} E(i) = \mathbb{1}, \tag{1}$$

Before defining the probability of a specific i occurring, we need the notion of a density matrix [**?**]:

Definition 3.3 (Density Matrix [**?** , Page 102])**.** *A density matrix ρ is a PSD matrix of trace one, i.e. $\mathrm{Tr}[\rho] = 1$.*

Definition 3.4 (Event Probability [**?** , Page 102])**.** *Let ρ be a density matrix and let i denote an event with $E(i)$ being the corresponding element from the POVM. The probability of the event i happening, i.e. measuring the outcome i, is given by*

$$p(i) = \mathrm{Tr}[\rho E(i)] \tag{2}$$

Proposition 3.5. *The expression in Equation 2 defines a valid probability distribution.*

Proof. While this is a well-known result we were not able to identify a concise proof in the literature and provide therefore, for the sake of completeness, a proof in Appendix B.1 □

From a formal perspective, POVMs constitute an elegant way of defining probability distributions. However, from a computational perspective the constraint that $\sum_i E(i) = \mathbb{1}$ causes some issues. While one could enforce the constraint using eigenvalue or singular value decompositions, performing these within a learning loop we would need to differentiate through the decomposition algorithms, which could lead to numerical stability issues. Especially, when computing gradients.[1] To circumvent these issues we (slightly) generalize Definition 3.4.

Definition 3.6 (Z-Normalized Event Probability)**.** *Let ρ be a PSD matrix and let i denote an event with $E(i)$ being the associated PSD matrix. The probability of the event i happening, i.e. measuring the outcome i, is given by*

$$\mathrm{Tr}\left[\rho E(i)\right] \big/ Z, \qquad \text{where } Z = \sum_{i=0}^{V-1} \mathrm{Tr}\left[\rho E(i)\right]. \tag{3}$$

[1] For instance, the PyTorch documentation mentions some of these concerns for singular value decompostions (link) and for eigenvalue value decompositions (link).

In comparison to Definition 3.4, Definition 3.6 does not only drop the requirement that $\sum_i E(i) = \mathbb{1}$ but also that $\mathrm{Tr}[\rho] = 1$.

Proposition 3.7. *The expression in Equation 3 defines a valid probability distribution.*

Proof. In order to prove this we need to show that $\forall : \mathrm{Tr}[E(i)\rho] \geq 0$. This then implies that $Z > 0$ (assuming that at least one event has non-zero probability). Note also that the constraint $\sum_i \mathrm{Tr}[E(i)\rho]/Z = 1$ is trivially satisfied by construction. Showing that $\forall : \mathrm{Tr}[E(i)\rho] \geq 0$ amounts to showing that the trace of a matrix-matrix product of two PSD matrices is positive real-valued. We already prove this as a sub-step in the proof for Proposition 3.5 (cf. Equation A4 in the appendix). This finishes the proof. □

4 Positive Operator Circuits

With the slightly generalized definition of an event probability at hand we are now ready to define positive operator circuits (using partition circuits).

Definition 4.1 (Positive Operator Circuit)**.** *Let $\mathbf{X} = \{X_0, \ldots, X_{M-1}\}$ (taking values $\mathbf{x} = \{x_0, \ldots, x_{M-1}\}$) be a set of categorical random variables of sample space size N. Furthermore, let B be a positive integer and let $\rho \in \mathbb{C}^{B\times B}$ be a PSD matrix. We define a positive operator circuit as a partition circuit whose computation units take the following functional form:*

$$p_k(\mathbf{x}_k) = \begin{cases} F_k \times e_{x_k} \otimes e^*_{x_k} \times F^*_k, \quad e_{x_k} \in \mathbb{C}^N, F_k \in \mathbb{C}^{B\times N} & \text{if } k \text{ leaf unit} \\ G_k \times \Big(p_{k_l}(\mathbf{x}_{k_l}) \odot p_{k_r}(\mathbf{x}_{k_r})\Big) \times G^*_k, \quad G_k \in \mathbb{C}^{B\times B} & \text{if } k \text{ internal unit} \\ \mathrm{Tr}\Big[\rho \times \Big(p_{k_l}(\mathbf{x}_{k_l}) \odot p_{k_r}(\mathbf{x}_{k_r})\Big)\Big] \Big/ Z, \quad \rho \in \mathbb{C}^{B\times B}, Z \in \mathbb{R}_{>0} & \text{if } k \text{ root unit.} \end{cases} \tag{4}$$

Here we use the symbols $\times$, $\otimes$, $\odot$ and $\cdot$ to denote the matrix product, Kronecker product, Hadamard product, respectively. Furthermore, $\{e_{x_n}\}_{n=0}^{N-1}$ is a set of N-dimensinoal orthonormal basis vectors associating each outcome of the random variable X_k to a specific basis vector. The symbol Z denotes the normalization constant for which we require:

$$Z = \sum_{\mathbf{x}\in\Omega(\mathbf{X})} \mathrm{Tr}\Big[\rho \times \Big(p_{root_l}(\mathbf{x}_{root_l}) \odot p_{root_r}(\mathbf{x}_{root_r})\Big)\Big] \tag{5}$$

Theorem 4.2. *POXs are valid probability distributions.*

Proof. We need to prove that at the root we have $p_{root}(\mathbf{x}) \geq 0$, for every $\mathbf{x} \in \Omega(\mathbf{X})$ and that $\sum_{\mathbf{x}\in\Omega(\mathbf{X}_{root})} p_{root}(\mathbf{x}) = 1$. The latter requirement is satisfied by construction via the normalization constant Z. For the former, let us first define:

$$\mu_{root}(\mathbf{x}) \coloneqq p_{root_l}(\mathbf{x}_l) \odot p_{root_r}(\mathbf{x}_r). \tag{6}$$

By identifying each element $\mathbf{x} \in \Omega(\mathbf{X})$ with an event i from Definition 3.6, consequently $\mu_{root}(\mathbf{x})$ with $E(i)$, we observe that the root of a POX has the functional form of the probability of an element from a (generalized) POVM. In order to prove that $\forall \mathbf{x} \in \Omega(\mathbf{X}) : \mathrm{Tr}[\mu_{root}(\mathbf{x})\rho] > 0$, we now only need to show that $\mu_{root}(\mathbf{x})$ is PSD, as ρ is already PSD by assumption. We show this in Appendix B.2. □

4.1 Discussion on Computational Complexity

In order to determine the computational complexity of evaluating a positive operator circuit, we simply study the cost of the individual computation units. We start at the leaves where we first have a Kronecker product $e_{x_k} \otimes e^*_{x_k}$, which can be performed in $\mathcal{O}(N^2)$. The matrix products with F_k and F^*_k can then be performed in $\mathcal{O}(BN^2)$. In the internal layers we perform the Hadamard product in $\mathcal{O}(B^2)$, followed again by two matrix-matrix products in $\mathcal{O}(B^3)$. In the root we compute the numerator of the fraction with a Hadamard product ($\mathcal{O}(B^2)$). The trace of the resulting matrix-matrix product can then be obtained in $\mathcal{O}(B)$. This means that evaluating a circuit (ignoring the normalization constant for now) has a cost of $\mathcal{O}(BN^2)$ per computation unit (assuming $B < N$). Furthermore, it is easy to show that partition circuits, where computation units have two inputs, have a total of $\mathcal{O}(M)$ computation units, with M being the number of input variables. We conclude that the computational complexity of evaluating a positive operator circuit is $\mathcal{O}(MBN^2)$.

In the discussion above we assumed that the normalization constant Z was given. Usually, this is a strong assumption as computing the normalization constant is in general computationally hard. However, for POXs we can, similarly to probabilistic circuits, exploit the decomposability property and compute Z in polytime via tractable marginalization.

Proposition 4.3. *POXs allow for tractable marginalization.*

Proof. The proof is trivial, as marginalizing out a variable X_m from a POX simply amounts to pushing down the summation to the corresponding leaf. This is possible as we can exchange at the root the summation and the trace, and exploit that the internal computation units are multilinear functions in terms of the input variables. At the leaf $p_m(x)$ we then have:

$$\sum_{x\in\Omega(X_m)} F_m \times e_x \otimes e_x^* \times F_m^* = F_m \times \left(\sum_{x\in\Omega(X_m)} e_x \otimes e_x^*\right) \times F_m^* = F_m \times \mathbb{1} \times F_m^* = F_m \times F_m^*, \tag{7}$$

where we exploit the fact that the e_n's are orthonormal basis vectors, which allows us to replace the sum with the identity matrix. □

If we marginalize out all variables $\mathbf{X}$ using the prescription described in the proof of Proposition 4.3, we obtain a circuit that does not dependent anymore on any inputs but which is still evaluable in time $\mathcal{O}(MBN^2)$ (using identical reasoning to the argument for the input dependent case). We conclude that we can compute Z in time polynomial in the size of the circuit and that positive operator circuits can be evaluated in $\mathcal{O}(MBN^2)$ time. We show next how to bring this down to $\mathcal{O}(MBN)$.

4.2 The Vector Representation of Positive Operator Circuits

The cubic computational dependency of the computation cost per computation unit $bigO(BN^2)$ is due to the presence of matrix-matrix multiplications. By rearranging the operations performed in the positive operator circuit we can avoid these matrix-matrix multiplications entirely and obtain matrix-vector multiplications instead. We call these alternative representation of POXs *positive vector circuits*.

Definition 4.4 (Positive Vector Circuit)**.** *Let $\mathbf{X} = \{X_0, \ldots, X_{M-1}\}$ (taking values $\mathbf{x} = \{x_0, \ldots, x_{M-1}\}$) be a set of categorical random variables of sample space size N and let $\{e_{x_n}\}_{n=0}^{N-1}$ be a set of orthonormal basis vectors with elements from the sample space being bijectively associated to basis vectors. Furthermore, let B be a positive integer and let $\rho \in \mathbb{C}^{B\times B}$ be a PSD matrix. We define a positive vector circuit as a partition circuit whose computation units take the following functional form:*

$$q_k(\mathbf{x}_k) = \begin{cases} F_k \times e_{x_k}, & \text{if } n \text{ leaf unit} \\ G_k \times \Big(q_{n_l}(\mathbf{x}_{k_l}) \odot q_{k_r}(\mathbf{x}_{k_r})\Big), & \text{if } k \text{ internal unit} \\ \nu_k(\mathbf{x}_k) \cdot \nu_k^*(\mathbf{x}_k), & \text{if } k \text{ root unit,} \end{cases} \tag{8}$$

where we have $\odot$, $\times$, and $\cdot$ denoting the Hadamard product, the matrix-vector product, and the dot product, respectively. Furthermore, we used:

$$\nu_{root}(\mathbf{x}_{root}) \coloneqq \gamma \times \Big(q_{k_l}(\mathbf{x}_{root_l}) \odot q_{k_r}(\mathbf{x}_{root_r})\Big) \big/ \sqrt{Z}, \tag{9}$$

with $\gamma \in \mathbb{C}^{B\times B}$ such that $\rho = \gamma \times \gamma^*$. The matrices F_k and G_k are defined as in Definition 4.1.

Following a similar train of thought as in Section 4.1 we can derive the computational cost of evaluating a positive vector circuit to be $\mathcal{O}(MBN)$ (assuming that the normalization constant has been pre-computed and which is a one-time cost). The major difference between positive operator circuits and positive vector circuits is that for the former the internal computation units output matrices ($p_k(\mathbf{x}_k) \in \mathbb{C}^{B\times B}$) while for the latter vectors are outputted ($q_k(\mathbf{x}_k) \in \mathbb{C}^B$).

Proposition 4.5. *The root nodes of positive operator circuits and positive vector circuits compute the same probability. That is, $p_{root}(\mathbf{x}) = q_{root}(\mathbf{x})$ if both circuits use the same F_k's, G_k's, and ρ.*

Proof. We show this in Appendix B.3. □

A similar observation and ensuing proof has been given by **?**], although for a more restricted setting. We describe the relationship to their work in more detail in Section 5.

4.3 Parametrizing Positive Operator Circuits

So far we have only given the computational structure of POXs and constraints that have to be satisfied for POXs to form valid probability distributions, e.g. positive semidefiniteness. A parametrization can be constructed as follows: at the root unit we need an (unnormalized) density matrix ρ, i.e. we need to construct a PSD matrix. As already hinted at in Definition 4.4 this is easily achieved by setting

$$\rho = \gamma \times \gamma^*, \tag{10}$$

where $\gamma \in \mathbb{C}^{B \times B}$ is an arbitrary $B \times B$ complex valued matrix.

For the parameter matrices F_k and G_k in the leaf and internal units we can simply pick arbitrary complex valued matrices with adequate dimensions. For the set of basis vectors $\{e_{x_n}\}_{n=0}^{N-1}$ in the leaves we simply choose the set of standard basis vectors, i.e. one-hot unit vectors, with each basis vector e_n corresponding to a random outcome. Although different choices of basis vectors would be valid as well, e.g. the columns of the orthonormalized discrete Fourier transform matrix.

Even though complex numbers are well-supported in modern deep learning libraries and the gradients can be computed with automated differentiation using Wirtinger calculus [**?** **?**], the issue of numerical stability with complex numbers has to be handled more carefully. Similar to computations with classic probabilities, computations with positive operator and vector circuits have to be performed in log-space.

A naive implementation of complex numbers would, similar to probabilities, quickly result in the absolute value r of a complex number $re^{i\phi}$ either under- or overflowing when performing repeated multiplications. Therefore, we represent complex-valued parameters using the polar form $re^{i\phi}$ in the log domain using a tuple of real parameters: $(\log r, \phi)$. In this representation ϕ and $\log r$ are both unconstrained and real-valued. Multiplications of two number can now be computed in a straightforward fashion by adding up the elements of the tuples: $(\log r_1, \phi_1) \times (\log r_2, \phi_2) = (\log r_1 + \log r_2, \phi_1 + \phi_2)$.

In order to perform complex-valued matrix-matrix and matrix-vector multiplications in the log-domain, we adapt the *LogEinsumExp-trick* (a generalization of the LogSumExp trick) introduced by **?**] for positive real-valued PCs to the setting of complex numbers. This generalizes log-domain circuit evaluations with only positive reals, as well as circuit evaluations that allow for negative reals [**?** **?**]. The main idea is to perform the LogSumExp trick on the magnitude of the complex number (in log-space) $\log r$ only, and leave the phase ϕ alone. We refer the reader to our implementation of POXs/PVXs for further details.

5 Related Work

This work is inspired by theoretical observations made in the statistical relational AI literature. Specifically, **?**], and **?**] noted that using only real-valued parametrizations (including negatives [**?**]) does not allow for fully expressive models. In contrast to our work, both of these works are not concerned with learning and are of a rather theoretical nature. We discuss next three approaches from the physics and the machine learning literature that have taken a more practical approach.

Tensor Networks

A popular technique to model systems in condensed matter physics are so-called tensor networks [**?** **?**], and in recent years they have been applied to tackle problems in supervised as well as unsupervised machine learning [**?** **?** **?**] – with the works of **?**] and **?**] on tree tensor networks being most related to POXs. Similar to POXs, tree tensor networks perform inference in a hierarchical fashion. The conceptual difference, however, lies with the formulation of POXs in the POVM formalism. In this regard and given that tensor networks originate in the physics community, it is rather surprising that tensor networks have so far, and to the best of our knowledge, not been formulated using POVMs.

Using our generalized POVM formulation for POXs is, however, not only theoretically elegant but has also practical benefits: using this formalism leads to circuits that are normalized by construction (using a normalization constant), and we can perform learning simply by optimizing the likelihood. We contrast this to the sweeping algorithms that are usually deployed in the tensor network literature,

where blocks of variables are optimized one at the time while the remaining variables are held constant. It appears that this approach is inspired by the variational ansatz taken in the density-matrix renormalization group algorithm [**?**], which is the original algorithm for tensor networks.

We would also like to point out a theoretical result from the tensor network literature stating that picking a complex-valued parametrization instead of a real-valued one can lead to an arbitrarily large reduction in the number of parameters [**?**]. While this result is formulated with respect to (exact) low rank tensor decomposition and does not apply directly to the problem of learning parameters via gradient descent, we consider this observation to be a strong theoretical indicator for the superiority of complex numbers over real numbers when parametrizing probabilistic circuits. A similar argument has also been made by **?**] in the context of hidden Markov models.

Squared Non-Monotonic Probabilistic Circuits

?] recently introduced a class of tractable probabilistic models called *squared non-monotonic PCs* (NPC^{2}s). The main idea behind NPC^{2}s is that one can parametrize a probabilistic circuit using negative and positive reals and simply square the final output. They then use the circuit squaring algorithm from [**?**] to guarantee tractable marginalization and compute the normalization constant in time polynomial in the size of the circuit. Squaring the output of a circuit in order to guarantee positivity and normalizing via a normalization constant is reminiscent of the positive vector circuits from Section 4.2.

Proposition 5.1. *NPC^{2}s are real-valued positive vector circuits with a density matrix of rank-one.*

Proof. We show this in Appendix B.4. □

Concerning the parametrization, the biggest difference is our use of the entire complex plane compared to real valued parameters only for NPC^{2}s. While we do not have a theoretical proof that this complex parametrization results in improved expressive power, results from the tensor network literature [**?**] and the statistical relation AI [**? ?**] literature hint in this direction.

Using Proposition 5.1 and the fact that NPC^{2}s only use real-valued parameters we can conclude that positive vector circuits generalize NPC^{2}s two-fold. First, PVXs use full-rank matrices as their (unnormalized) density matrix. Second, PVXs utilize the entire complex plane. In our experiments we show that these generalizations do have an impact. Interestingly the restriction to rank-one density matrices is also customary in the tensor network literature [**? ?**].

Positive Semidefinite Kernels

Recently, the use of PSD matrices has also been studied in the kernel literature [**? ?**]. Specifically, given a set of N data samples $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(N-1)}$. An unnormalized probability distribution $q(\mathbf{x})$ can be defined as follows: $q(\mathbf{x}) = \boldsymbol{\kappa}^T(\mathbf{x}) A \boldsymbol{\kappa}(\mathbf{x})$, where $A \in \mathbb{R}^{N \times N}$ is a PSD matrix and where $\boldsymbol{\kappa}(\mathbf{x}) \in \mathbb{R}^N$ is a vectorized kernel: $\boldsymbol{\kappa}_i(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{x}^i)$ and κ being the base kernel, e.g. radial basis function. Comparing this to the root computation unit of a PVX, we can observe a superficial resemblance: in both cases we have a quadratic form. Important differences exist, however. Contrary to PVXs the size of the PSD kernels grows with the number of data points. Furthermore, the distribution induced by PSD kernels is unnormalized.

6 Experimental Evaluation

Experimental Setup

For our experimental evaluation we used the MNIST family of datasets. That is, the original MNIST [**?**], FashionMNIST [**?**], and also EMNIST [**?**]. For the implementation we used PyTorch together with the Einops library [**?**]. We set up our experiments in Lightning[2] and ran them on a DGX-2 machine with V100 Nvidia cards.

We compare different methods using the *bits per dimension* metric, which is calculated from the average negative log-likelihood ($\overline{NLL}$) as follows: $bpd = \overline{NLL} / (\log 2 \times D)$ ($D = 28^2$ for MNIST datasets).

[2] https://lightning.ai/

Training Positive Operator Circuits

In our experiments we estimate the density of the different MNIST datasets by maximizing a parametrized likelihood function. We construct this function using a positive vector circuit. In order to build the underlying partition circuit, we follow the approach described by **?**]: we start with a grid of 28×28 pixels and merge, in an alternating fashion, rows and columns of the image. We also allow for merging three partitions into a single partition, thereby breaking the binary character of the partition trees used so far. Having three instead of two partitions being merged can easily be accommodated for by using a Hadamard product with three factors instead of two in the internal units of a PVX. Ultimately, this allows us to handle layers with an uneven number of partitions. At the leaves we encode pixel values by associating each value to one of 256 standard basis vector of $\mathbb{R}^{256}$.

We performed training by minimizing the negative log-likelihood using Adam [**?**] with default hyperparameters, batch size of 50, over 100 epochs. The best model was selected using a $90 - 10$ train-validation data split where we monitored the negative log-likelihood on the validation set. We did not perform any hyperparameter search. Further details can be found in the configuration files of the experiments.

State-of-the-Art Baselines

We compare PVXs to three state-of-the-art tractable density estimators from the literature: quadrature of probabilistic circuits (QPCs) [**?**], hidden Chow-Liu trees (HCLTs) [**?**], and sparse HCLTs (SHCLTs) [**?**]. All three methods are probabilistic circuits and they all use hidden Chow-Liu trees as their underlying structure. This tree is learned and dataset specific. The difference between QPCs and HCLTs is that the former is obtained by approximating a continuous latent variable extension of probabilistic circuits using numerical quadrature. The difference between HCLTs and SHCLTs is that the latter allows for dynamically adapting the number of components per partition during parameter learning. This means that SHCLTs can focus their computational budget on information-rich parts of the circuit.

Q1: How Do the Different PVX Parametrizations Fair Against Each Other?

We construct PVXs following the prescription of [**?**], as described above. Furthermore, we use $B = 128$ components per partition. As for the parametrization we study four different variants: PVX_{0}^{FR}, $\mathrm{PVX}_{2\pi}^{FR}$, PVX_{0}^{R1}, and $\mathrm{PVX}_{2\pi}^{R1}$. The subscripts (0 or 2π) indicate whether we limit the phase to be zero or whether we allow the phase to be a learnable parameter. Note that having a tunable phase doubles the number of (real-valued) parameters. The superscript indicates whether the parametrization uses a full-rank (FR) density matrix in the root or a rank-one ($R1$) density matrix. As such, PVX_{0}^{R1} is equivalent to the squared monotonic probabilistic circuits (MPC^{2}s) used in [**?**], and $\mathrm{PVX}_{2\pi}^{R1}$ generalizes their NPC^{2}s from the real domain to the entire complex domain. Concretely, MPC^{2}s are rank-one PVXs with the phase fixed to zero and NPC^{2}s are rank-one PVXs with the phase fixed to values in the set $\{0, \pi\}$. We do not experiment on the latter.

We report the obtained bpds for the different PVX parametrizations in the first four columns of Table 1. In general, we see that the complex-valued parametrizations are either on par or outperform the corresponding zero-phase parametrizations. The notable exception is the FashionMNIST benchmark where the two parametrizations with no phase (PVX_{0}^{FR} and PVX_{0}^{R1}) perform best. Comparing full-rank to rank-one parametrizations we see a more important impact for the zero-phase parametrization than for the complex-valued parametrization.

Overall, $\mathrm{PVX}_{2\pi}^{FR}$ constitutes the best performing parametrization being best-in-class on all benchmarks but FashionMNIST.

Q2: How Do PVXs Fair Against the State of the Art?

In order to compare PVXs to state-of-the-art circuits all methods (PVX, QPC, HCLT, SHCLT) were given a computational budget of $B = 128$ components per partition (on average for SHCLT). The results for the competing methods were taken from the respective papers – except for HCLTs. For HCLTs we took the bpds reported by **?**] as they achieved stronger results with their implementation compared to the originally reported performance [**?**].

We see in Table 1 that QPCs and HCLTs are in general outperformed by PVX– in particular $\mathrm{PVX}_{2\pi}^{FR}$. The FashionMNIST benchmark constitutes again an outlier in this regard. The only methods outperforming PVXs are the SHCLTs, which is due to them being able to dynamically allocate

Table 1: Test set bpd for MNIST datasets (lower is better).

	PVX_0^{FR}	$\mathrm{PVX}_{2\pi}^{FR}$	PVX_0^{R1}	$\mathrm{PVX}_{2\pi}^{R1}$	QPC	HCLT	SHCLT
MNIST	1.16	1.16	1.24	1.17	1.18	1.21	1.14
FashionMNIST	3.37	3.55	3.44	3.55	3.27	3.34	3.27
E-MNIST	1.69	1.63	1.76	1.64	1.66	1.70	1.52
E-LETTERS	1.62	1.60	1.70	1.61	1.70	1.75	1.58
E-BALANCED	1.65	1.64	1.73	1.64	1.73	1.78	1.60
E-BYCLASS	1.47	1.47	1.56	1.49	1.67	1.73	1.54

compute budget to informative parts of the circuit. It is noteworthy that for the EMNIST-BYCLASS benchmark all four PVX parametrization exhibit strong performance with $\mathrm{PVX}_{2\pi}^{FR}$ and $\mathrm{PVX}_{2\pi}^{R1}$ even outcompeting SHCLTs.

Discussion

In our experimental evaluation we did not perform an explicit comparison to the method of **?**] as they were not able to find any improvements of allowing non-negative parameters in probabilistic circuit for discrete data. They stipulated that "simple categorical distribution can already capture any discrete distribution with finite support and a (subtractive) mixture thereof might not yield additional benefits". In our experimental evaluation we refute this conjecture, and show that using complex-valued parameters leads to noticeable gains when performing density estimation on discrete data. The exception being of course the FashionMNIST benchmark. We believe that this is due to our optimization method that was not tuned in any way. In this regard we believe that developing tailor-made optimization algorithms for PVXs might further boost their performance. Furthermore, one can envisage, similar to SHCLTs, dynamically adapting the compute budget per partition, which should again improve the density estimation capacities of PVXs.

7 Conclusions & Future Work

Based on first principles from quantum information theory, we constructed positive operator circuits – a novel class of probabilistic tractable models: Their construction as partition circuits allows for layer-wise parallelization and execution on modern machine learning hardware. Using unconstrained gradient-descent we showed that POXs, and PVXs specifically, constitute a promising addition to the zoo of tractable probabilistic models.

In future work we would like to investigate in more detail theoretical properties of POXs and how they differ from other tractable models. Ideally one would find an exponential separation between real-valued and complex-valued circuits. On the practical side it remains, however, to be seen whether such a separation has an equally important impact on real-world datasets.

This relates to another open question, that of learning in POXs. We presented a rather simple learning approach for parameters tailored towards neural architecture. It might be the case that more sophisticated techniques have to be deployed for large-scale POXs, as they might exhibit the problem of barren plateaus [**?**] – a well known issue in quantum machine learning [**?** **?**]. Furthermore, and as already pointed out by **?**], computing the normalization constant Z requires the evaluation of the circuit in the POX representation. This gives rise to a rather expensive cubic computation cost during learning (when compared to the quadratic cost of PVX evaluations). Avoiding this issue would allow to drastically scale PVXs. In order, to scale PVXs it might also be necessary to use more sophisticated structures than binary trees with every partition having the same number of components B per partition, cf. SHCLTs.

A Probabilistic Circuits as Partition Circuits

Definition A.1 (Layered Probabilistic Circuit)**.** *Let $\mathbf{X} = \{X_0, \ldots, X_{M-1}\}$ (taking values $\mathbf{x} = \{x_0, \ldots, x_{M-1}\}$) be a set of categorical random variables. Furthermore, let B be a positive integer. We define a layered probabilistic circuit as a partition circuit whose computation units take the following functional form:*

$$p_k(\mathbf{x}_k) = \begin{cases} f_k(x_k), \quad f_k(x_k) \in \mathbb{R}^B_{\geq 0} & \text{if } n \text{ leaf unit, i.e. } \mathbf{x}_k = \{x_k\} \\ W_k \times \Big(p_{n_l}(\mathbf{x}_{n_l}) \odot p_{n_r}(\mathbf{x}_{n_r})\Big), \quad W_k \in \mathbb{R}^{B \times B}_{\geq 0} & \text{if } n \text{ internal unit} \\ \rho \cdot \Big(p_{n_l}(\mathbf{x}_{n_l}) \odot p_{n_r}(\mathbf{x}_{n_r})\Big)/Z, \quad \rho \in \mathbb{R}^B_{\geq 0} & \text{if } n \text{ root unit.} \end{cases} \tag{A1}$$

Here we use the symbols $\times$, $\odot$ and $\cdot$ to denote the matrix product, Hadamard product and dot product, respectively. Additionally, we necessitate:

$$Z = \sum_{\mathbf{x} \in \Omega(\mathbf{X})} p_{root}(\mathbf{x}) \tag{A2}$$

where $\Omega(\mathbf{X})$ denotes the sample space of $\mathbf{X}$.

Theorem A.2. *Layered PCs are valid probability distributions.*

Proof. If $p_{root}(\mathbf{x})$ is the computation unit at the root of the layered PC, the unit forms a probability distribution if $p(x) \geq 0$, for every $\mathbf{x} \in \Omega(\mathbf{X})$ and if $\sum_{\mathbf{x} \in \Omega(\mathbf{X})} p(\mathbf{x}) = 1$. The first condition is trivially satisfied as the circuit only performs linear operations on matrices and vectors $(f_k(x_k), W_k, \rho)$ with positive entries only. The second requirement is satisfied by construction using the normalization constant Z. □

B Proof of Theorems and Propositions

B.1 Proof of Proposition 3.5

Proposition 3.5. *The expression in Equation 2 defines a valid probability distribution.*

Proof. First we show that $p(i) \geq 1$, for each i:

$$p(i) = \mathrm{Tr}[E(i)\rho] = \mathrm{Tr}[F(i)F^*(i)\rho] = \mathrm{Tr}[F(i)^*\rho F(i)]. \tag{A3}$$

Here we used the fact that $E(i)$ is PSD and factorized it into the product $F(i)F^*(i)$. Then we used the fact that the trace is invariant under cyclical shifts. Next we write out the trace using matrix elements:

$$\mathrm{Tr}[F(i)^*\rho F(i)] = \sum_{klmn} \delta_{kn} F^*_{lk}(i)\rho_{lm}F_{mn}(i) = \sum_k F^*_k(i)\rho F_k(i). \tag{A4}$$

In the equation above $F_k(i)$ is the k-th column of $F(i)$. As ρ is PSD we have that each term of the sum over k is positive, which means in turn that $p(i)$ is a positive real number, i.e. $p(i) \geq 0$ for every i.

Secondly, we show that $p(i)$ is normalized:

$$\sum_{i=1}^N p(i) = \sum_{i=1}^N \mathrm{Tr}[E(i)\rho] = \mathrm{Tr}\left[\sum_{i=1}^N E(i)\rho\right] = \mathrm{Tr}[\rho], \tag{A5}$$

where we used Equation 1. Exploiting the fact that the trace of a density matrix is one gives us indeed $\sum_{i=1}^N p(i) = 1$, and we can conclude that $p(i)$ is a valid probability distribution. □

B.2 Proof of Theorem 4.2

Theorem 4.2. *POXs are valid probability distributions.*

Proof. We need to prove that at the root we have $p_{root}(\mathbf{x}) \geq 0$, for every $\mathbf{x} \in \Omega(\mathbf{X}_{root})$ and that $\sum_{\mathbf{x} \in \Omega(\mathbf{X}_{root})} p_{root}(\mathbf{x}) = 1$. The latter requirement is satisfied by construction via the normalization constant Z. For the former, let us first define:

$$\mu_{root}(\mathbf{x}) \coloneqq p_{root_l}(\mathbf{x}_l) \odot p_{root_r}(\mathbf{x}_r). \tag{A6}$$

By identifying each element $\mathbf{x} \in \Omega(\mathbf{X})$ with an event i from Definition 3.6, consequently $\mu_{root}(\mathbf{x})$ with $E(i)$, we observe that the root of a POX has the functional form of the probability of an element from a (generalized) POVM. In order to prove that $\forall \mathbf{x} \in \Omega(\mathbf{X}) : \mathrm{Tr}[\mu_{root}(\mathbf{x})\rho] > 0$, we now only need to show that $\mu_{root}(\mathbf{x})$ is PSD, as ρ is already PSD by assumption.

To this end, we first observe that in the leaf units of a POX we have the cross product between a vector and its conjugate transpose, which produces a PSD matrix. These PSD matrices from the leaves are then fed into the first layer of internal units where we first perform Hadamard products, which is again a PSD matrix via Schur's product theorem [**?** , p. 14, Theorem VII]. For units k in the first internal layer we then have:

$$p_k(\mathbf{x}_k) = G_k \times P_k(\mathbf{x}_k) \times G_k^*, \tag{A7}$$

where $P_k(\mathbf{x}_k)$ is the matrix obtained by performing the Hadamard product in the leaf. Using the fact that $P_k(\mathbf{x})$ is PSD allows for a square root decomposition, and we can write:

$$p_k(\mathbf{x}_h) = G_k P_k^{1/2}(\mathbf{x}_n) P_k^{1/2}(\mathbf{x}_k) G_k^* = \left(G_k P_k^{1/2}(\mathbf{x}_k)\right)\left(G_k P_k^{1/2}(\mathbf{x}_k)\right)^*, \tag{A8}$$

which is clearly PSD. By repeating this argument recursively for all the layers up to the root we can conclude that $p_{root}(\mathbf{x})$ is PSD for all $\mathbf{x} \in \Omega(\mathbf{X}_{root})$, which also concludes the proof. □

B.3 Proof of Proposition 4.5

Proposition 4.5. *The root nodes of positive operator circuits and positive vector circuits compute the same probability. That is, $p_{root}(\mathbf{x}) = q_{root}(\mathbf{x})$ if both circuits use the same F_k's, G_k's, and ρ.*

Proof. We start the proof by trivially rewriting the computation units in the leaves of a POX (Equation 4):

$$p_k(x_k) = (F_k \times e_{x_k}) \otimes (F_k \times e_{x_k})^* = q_k(x_k) \otimes q_k^*(x_k), \tag{A9}$$

This means that we can construct the matrix $p_k(\mathbf{x}_k)$ from the vector $q_k(\mathbf{x}_k)$, and we need only to pass on the vector to the next layer. Coincidentally, $q_k(\mathbf{x}_k)$ is the exact computation performed by the PVX at unit k. This also means that we only need to perform the computation for $q_k(\mathbf{x}_k)$ and not its conjugate transpose.

In the next layer, we plug in the right-most side of Equation A9 into the expression of an internal computational unit of a POX and we get:

$$p_k(\mathbf{x}_k) = G_k \times \left[\left(q_{k_l}(\mathbf{x}_{k_l}) \otimes q_{k_l}^*(\mathbf{x}_{k_l})\right) \odot \left(q_{k_r}(\mathbf{x}_{k_r}) \otimes q_{k_r}^*(\mathbf{x}_{k_r})\right)\right] \times G_k^*. \tag{A10}$$

We now exploit the mixed-product property of the Hadamard product and the Kronecker product to obtain:

$$\begin{aligned} p_k(\mathbf{x}_k) &= G_k \times \left[q_{k_l}(\mathbf{x}_{k_l}) \odot q_{k_r}(\mathbf{x}_{k_r})\right) \otimes \left(q_{k_l}^*(\mathbf{x}_{k_l}) \odot q_{k_r}^*(\mathbf{x}_{k_r})\right)\right] \times G_k^* && \text{(A11)} \\ &= \left[G_k \times \left(q_{k_l}(\mathbf{x}_{k_l}) \odot q_{k_r}(\mathbf{x}_{k_r})\right)\right] \otimes \left[G_k \times \left(q_{k_l}(\mathbf{x}_{k_l}) \odot q_{k_r}(\mathbf{x}_{k_r})\right)\right]^* && \text{(A12)} \\ &= q_k(\mathbf{x}_k) \otimes q_k^*(\mathbf{x}_k). && \text{(A13)} \end{aligned}$$

Going from Equation A11 to Equation A12 we made use of associativity.

We can now recursively plug in in these Kronecker decompositions (cf. Equation A13) into subsequent computation units and will obtain for each internal layer the one-to-one correspondence of:

$$p_k(\mathbf{x}_k) = q_k(\mathbf{x}_k) \otimes q_k^*(\mathbf{x}_k), \tag{A14}$$

which means that we can evaluate the internal units of a POX by evaluating the corresponding internal units in the PVX.

Finally, at the root layer we plug in again the Kronecker decomposition from the last internal layer and obtain:

$$\begin{aligned} p_k(\mathbf{x}_k) &= \mathrm{Tr}\left[\rho \times \left(q_{k_l}(\mathbf{x}_{k_l}) \otimes q_{k_l}^*(\mathbf{x}_{k_l})\right) \odot \left(q_{k_r}(\mathbf{x}_{k_r}) \otimes q_{k_r}^*(\mathbf{x}_{k_r})\right)\right] \Big/ Z && \text{(A15)} \\ &= \mathrm{Tr}\left[\rho \times \left(q_{k_l}(\mathbf{x}_{k_l}) \odot q_{k_r}(\mathbf{x}_{k_r})\right) \otimes \left(q_{k_l}^*(\mathbf{x}_{k_l}) \odot q_{k_r}^*(\mathbf{x}_{k_r})\right)\right] \Big/ Z && \text{(A16)} \\ &= \mathrm{Tr}\left[\gamma^* \times \gamma \times \left(q_{k_l}(\mathbf{x}_{k_l}) \odot q_{k_r}(\mathbf{x}_{k_r})\right) \otimes \left(q_{k_l}^*(\mathbf{x}_{k_l}) \odot q_{k_r}^*(\mathbf{x}_{k_r})\right)\right] \Big/ Z && \text{(A17)} \\ &= \mathrm{Tr}\left[\gamma \times \left(q_{k_l}(\mathbf{x}_{k_l}) \odot q_{k_r}(\mathbf{x}_{k_r})\right) \otimes \left(q_{k_l}^*(\mathbf{x}_{k_l}) \odot q_{k_r}^*(\mathbf{x}_{k_r})\right) \times \gamma^*\right] \Big/ Z. && \text{(A18)} \end{aligned}$$

We can write this expression now easily in terms of ν_k

$$p_k(\mathbf{x}_k) = \mathrm{Tr}\big[\nu_k(\mathbf{x}_k) \otimes \nu_k^*(\mathbf{x}_k)\big] \tag{A19}$$
$$= \nu_k(\mathbf{x}_k) \cdot \nu_k^*(\mathbf{x}_k), \tag{A20}$$

which proves the equality of a POX and its corresponding PVX at the root. □

B.4 Proof of Proposition 5.1

Proposition 5.1. *NPC^{2}s are real-valued positive vector circuits with a density matrix of rank-one.*

Proof. To see this consider the computation at the root of a PVX:

$$\nu_{root}(\mathbf{x}) = \gamma \times \Big(q_{root_l}(\mathbf{x}_l) \odot q_{root_r}(\mathbf{x}_r)\Big)\Big/\sqrt{Z} \tag{A21}$$

Let us first define $\eta_{root}(\mathbf{x}) \coloneqq \Big(q_{root_l}(\mathbf{x}_l) \odot q_{root_r}(\mathbf{x}_r)\Big)\Big/\sqrt{Z}$. This then gives us:

$$\nu_{root}(\mathbf{x}) = \gamma \times \eta_{root}(\mathbf{x}) \tag{A22}$$

Here γ is an arbitrarily complex-valued matrix – potentially full-rank. If we, however, restrict γ to be a rank-one matrix, i.e. $\gamma = \delta \otimes \delta^*$, with δ being a complex-valued vector, we get:

$$\nu_{root}(\mathbf{x}) = (\delta \otimes \delta^*) \times \eta_{root}(\mathbf{x}). \tag{A23}$$

Plugging this into the expression for q_{root} (cf. Equation 9) we obtain:

$$q_{root}(\mathbf{x}) = \Big[(\delta \otimes \delta^*) \times \eta_{root}(\mathbf{x})\Big] \cdot \Big[(\delta \otimes \delta^*) \times \eta_{root}(\mathbf{x})\Big]^* \tag{A24}$$

Slightly rearranging the factors gives us:

$$q_{root}(\mathbf{x}) = \Big[\delta \times \underbrace{(\delta^* \cdot \eta_{root}(\mathbf{x}))}_{=:\epsilon_{root}(\mathbf{x})}\Big] \cdot \Big[\delta \times (\delta^* \cdot \eta_{root}(\mathbf{x}))\Big]^* \tag{A25}$$

As $\delta^* \cdot \eta_{root}(\mathbf{x})$ is a simple dot product between two vectors $\epsilon_{root}(\mathbf{x})$ must be a scalar. This lets us further rearrange factors to result in:

$$q_{root}(\mathbf{x}) = \epsilon_{root}(\mathbf{x})\epsilon^*_{root}(\mathbf{x})(\delta \cdot \delta^*) \tag{A26}$$

Here, the dot product $\delta \cdot \delta^*$ between two vectors results again in a scalar, which is also positive. We absorb this scalar for simplicity into $\epsilon_{root}(\mathbf{x})$ with $\widehat{\epsilon}_{root}(\mathbf{x}) = \epsilon_{root}(\mathbf{x})\sqrt{\delta \cdot \delta^*}$. Lastly, we can compute the probability using the product of two scalars:

$$q_{root}(\mathbf{x}) = \widehat{\epsilon}_{root}(\mathbf{x})\widehat{\epsilon}^*_{root}(\mathbf{x}), \tag{A27}$$

As **?**] only formulated NPC^{2}s for real-valued parameters, we need to impose a further assumption: $\widehat{\epsilon}_{root}(\mathbf{x}) = \widehat{\epsilon}^*_{root}(\mathbf{x})$, which only holds for strictly real-valued scalars. Finally, we can write the probability computed by a PVX as the square of a scalar:

$$q_{root}(\mathbf{x}) = \widehat{\epsilon}^2_{root}(\mathbf{x}) \tag{A28}$$

We conclude that we recover NPC^{2}s as a special case of PVX by assuming γ and consequently ρ to be a rank-one matrix, and furthermore assuming that we only have real-valued parameters. □