Supplemental Material

A. Proofs of Section 3

Proposition A.1. *The covariance matrix C satisfies:*

$$C = C^T\,, \quad C > 0\,, \quad C + iJ \geq 0\,. \tag{54}$$

Proof. We give the proof for $|\mathcal{X}| = 1$ and refer the reader to (Simon et al., 1987) for the general case. Suppressing the x index and denoting $\langle\cdot\rangle = \langle\psi|\cdot|\psi\rangle$, we have:

$$C = \begin{pmatrix} 2\sigma^2_{\widehat{\varphi}} & \langle\widehat{\varphi}\widehat{\pi} + \widehat{\pi}\widehat{\varphi}\rangle - 2\langle\widehat{\varphi}\rangle\langle\widehat{\pi}\rangle \\ \langle\widehat{\varphi}\widehat{\pi} + \widehat{\pi}\widehat{\varphi}\rangle - 2\langle\widehat{\varphi}\rangle\langle\widehat{\pi}\rangle & 2\sigma^2_{\widehat{\pi}}\,, \end{pmatrix} \tag{55}$$

where $\sigma^2_{\widehat{\varphi}} = \langle\widehat{\varphi}^2\rangle - \langle\widehat{\varphi}\rangle^2$ and similarly for $\widehat{\pi}$. For $|\mathcal{X}| = 1$, $C + iJ \geq 0$ is equivalent to $\mathrm{Det}(C + iJ) \geq 0$ or

$$4\sigma^2_{\widehat{\varphi}}\sigma^2_{\widehat{\pi}} \geq (\langle\widehat{\varphi}\widehat{\pi} + \widehat{\pi}\widehat{\varphi}\rangle - 2\langle\widehat{\varphi}\rangle\langle\widehat{\pi}\rangle)^2 + 1\,. \tag{56}$$

This is the uncertainty relation in the stronger form due to Robertson–Schroedinger, proving the statement for $|\mathcal{X}| = 1$. □

Proposition A.2. *The unitaries of* (20) *and* (21) *represent symplectic affine transformations of the canonical operators*

$$\widehat{D}(\boldsymbol{\xi})^\dagger \widehat{\boldsymbol{R}} \widehat{D}(\boldsymbol{\xi}) = \widehat{\boldsymbol{R}} + \boldsymbol{\xi} \tag{57}$$

$$\widehat{\omega}(S)^\dagger \widehat{\boldsymbol{R}} \widehat{\omega}(S) = S\widehat{\boldsymbol{R}}\,. \tag{58}$$

Proof. The result then follows from the Baker-Campbell-Hausdorff formula

$$\mathrm{e}^A B \mathrm{e}^{-A} = \mathrm{e}^{[A,\cdot]}(B) = B + [A,B] + \tfrac{1}{2}[A,[A,B]] + \cdots \tag{59}$$

and the commutators:

$$[i\widehat{\boldsymbol{R}}^T J\boldsymbol{\xi}, \widehat{R}_i] = -\xi_i\,, \quad [-\frac{i}{2}\widehat{\boldsymbol{R}}^T JX\widehat{\boldsymbol{R}}, \widehat{R}_i] = (X\widehat{\boldsymbol{R}})_i\,.$$

□

Proposition A.3. *Under the unitaries of* (20) *and* (21)*, the Gaussian states transform as*

$$\widehat{D}(\boldsymbol{\xi})\,|\boldsymbol{m}, C\rangle = |\boldsymbol{m} + \boldsymbol{\xi}, C\rangle \tag{60}$$

$$\widehat{\omega}(S)\,|\boldsymbol{m}, C\rangle = \left|S\boldsymbol{m}, SCS^T\right\rangle\,. \tag{61}$$

Proof. From proposition 3.3 and the definitions (16) and (17), we have for any $|\Psi\rangle$:

$$\langle\Psi|\,\widehat{D}(\boldsymbol{\xi})^\dagger \widehat{\boldsymbol{R}} \widehat{D}(\boldsymbol{\xi})\,|\Psi\rangle = \langle\Psi|\,(\widehat{\boldsymbol{R}} + \boldsymbol{\xi})\,|\Psi\rangle = \boldsymbol{m} + \boldsymbol{\xi}$$

$$\langle\Psi|\,\widehat{\omega}(S)^\dagger \widehat{\boldsymbol{R}} \widehat{\omega}(S)\,|\Psi\rangle = S\boldsymbol{m} = \boldsymbol{m}'$$

$$\langle\Psi|\,\widehat{\omega}(S)^\dagger \tfrac{1}{2}(\widehat{R}_i\widehat{R}_j + \widehat{R}_j\widehat{R}_i)\widehat{\omega}(S)\,|\Psi\rangle - m'_i m'_j = \sum_{k,l} S_{ik}\tfrac{1}{2}C_{kl}S_{jl} = \tfrac{1}{2}(SCS^T)_{ij}\,.$$

The result follows by specifying $|\Psi\rangle = |\boldsymbol{m}, C\rangle$. □

B. Proofs of Section 4

Proposition B.1. *The ODEs* (32)*,* (33) *have solutions*

$$\widehat{\varphi}_{x,a}(t) = F^{-1}\left(F(\widehat{\varphi}_{x,a}(0)) + t\right)\,, \tag{62}$$

$$\widehat{\pi}_{x,a}(t) = \frac{1}{2}\left(\widehat{\pi}_{x,a}(0)\frac{f(\widehat{\varphi}_{x,a}(0))}{f(\widehat{\varphi}_{x,a}(t))} + h.c.\right) \tag{63}$$

where $F'(x) = 1/f(x)$.

Proof. We can check directly the formulas by differentiating w.r.t. t to show that the time evolved fields satisfy the equation of motions. Rewriting the first equation as $F(\widehat{\varphi}_{x,a}(t)) = F(\widehat{\varphi}_{x,a}(0)) + t$ and differentiating the l.h.s.:

$$\partial_t F(\widehat{\varphi}_{x,a}(t)) = F'(\widehat{\varphi}_{x,a}(t))\dot{\widehat{\varphi}}_{x,a}(t) = \frac{\dot{\widehat{\varphi}}_{x,a}(t)}{f(\widehat{\varphi}_{x,a}(t))}\,, \tag{64}$$

which equals $\partial_t(F(\widehat{\varphi}_{x,a}(0)) + t) = 1$, showing that $\widehat{\varphi}_{x,a}(t)$ satisfies (32). For the second equation we differentiate the first term in the parenthesis:

$$\widehat{\pi}_{x,a}(0) f(\widehat{\varphi}_{x,a}(0))\partial_t\left(f(\widehat{\varphi}_{x,a}(t))\right)^{-1} \tag{65}$$

$$= -\widehat{\pi}_{x,a}(0)\frac{f(\widehat{\varphi}_{x,a}(0))}{f(\widehat{\varphi}_{x,a}(t))}\frac{f'(\widehat{\varphi}_{x,a}(t))}{f(\widehat{\varphi}_{x,a}(t))}\dot{\widehat{\varphi}}_{x,a}(t) \tag{66}$$

$$= -\widehat{\pi}_{x,a}(t) f'(\widehat{\varphi}_{x,a}(t))\,, \tag{67}$$

which shows that $\widehat{\pi}_{x,a}(t)$ satisfies (33). □

C. Proofs of Section 6

Lemma C.1. *Let $|\boldsymbol{m}, C\rangle$ be a Gaussian state and $\widehat{U}$ a unitary such that:*

$$\widehat{U}^\dagger \widehat{\boldsymbol{\varphi}} \widehat{U} = F(\widehat{\boldsymbol{\varphi}})\,. \tag{68}$$

Then:

$$|\,\langle\boldsymbol{\varphi}|\,\widehat{U}\,|\boldsymbol{m}, C\rangle\,|^2 = \left(F\#\mathcal{GP}(\boldsymbol{m}^1, C^{11})\right)(\boldsymbol{\varphi}) \tag{69}$$

where $f\#p$ denotes the push forward of p under f.

Proof. First we have:

$$|\,\langle\boldsymbol{\varphi}|\boldsymbol{m}, C\rangle\,|^2 = \mathcal{GP}(\boldsymbol{m}^1, C^{11})(\boldsymbol{\varphi})\,. \tag{70}$$

Now we note the representation of a projector as the average:

$$|\boldsymbol{\varphi}\rangle\,\langle\boldsymbol{\varphi}| = \int \frac{D\boldsymbol{\lambda}}{(2\pi)^{|\mathcal{X}|}}\mathrm{e}^{i\boldsymbol{\lambda}^T(\widehat{\boldsymbol{\varphi}} - \boldsymbol{\varphi})}\,, \tag{71}$$

a formula which can be proved by comparing matrix elements of the two operators in arbitrary states. Then we

have:

$$|\langle\boldsymbol{\varphi}|\,\widehat{U}\,|\boldsymbol{m},C\rangle|^2 = \langle\boldsymbol{m},C|\,\widehat{U}^\dagger\,|\boldsymbol{\varphi}\rangle\,\langle\boldsymbol{\varphi}|\,\widehat{U}\,|\boldsymbol{m},C\rangle \quad (72)$$

$$= \int \frac{D\boldsymbol{\lambda}\mathrm{e}^{-i\boldsymbol{\lambda}^T\boldsymbol{\varphi}}}{(2\pi)^{|\mathcal{X}|}}\,\langle\boldsymbol{m},C|\,\widehat{U}^\dagger\mathrm{e}^{i\boldsymbol{\lambda}^T\widehat{\boldsymbol{\varphi}}}\widehat{U}\,|\boldsymbol{m},C\rangle \quad (73)$$

$$= \int \frac{D\boldsymbol{\lambda}\mathrm{e}^{-i\boldsymbol{\lambda}^T\boldsymbol{\varphi}}}{(2\pi)^{|\mathcal{X}|}}\,\langle\boldsymbol{m},C|\,\mathrm{e}^{i\boldsymbol{\lambda}^T F(\widehat{\boldsymbol{\varphi}})}\,|\boldsymbol{m},C\rangle \quad (74)$$

$$= \int D\boldsymbol{\varphi'}\int \frac{D\boldsymbol{\lambda}\mathrm{e}^{i\boldsymbol{\lambda}^T(F(\boldsymbol{\varphi'})-\boldsymbol{\varphi})}}{(2\pi)^{|\mathcal{X}|}}|\,\langle\boldsymbol{\varphi'}|\boldsymbol{m},C\rangle\,|^2 \quad (75)$$

$$= \int D\boldsymbol{\varphi'}\delta(F(\boldsymbol{\varphi'})-\boldsymbol{\varphi})\mathcal{GP}(\boldsymbol{m}^1,C^{11})(\boldsymbol{\varphi'}) \quad (76)$$

$$= \left(F\#\mathcal{GP}(\boldsymbol{m}^1,C^{11})\right)(\boldsymbol{\varphi})\,. \quad (77)$$

In going from the third to the fourth line we used the following representation, valid for any f:

$$f(\widehat{\boldsymbol{\varphi}}) = \int D\boldsymbol{\varphi} f(\boldsymbol{\varphi})\,|\boldsymbol{\varphi}\rangle\,\langle\boldsymbol{\varphi}|\ , \quad (78)$$

and we have used the definition of push forward of a distribution, i.e. that if $x \sim p_X$, then $p_Z = f\#p_X$ is the distribution of the random variable $z = f(x)$ which can be obtained explicitly via the change of variable formula:

$$p_Z(z) = |\mathrm{Det}(\tfrac{\partial f(x)}{\partial x}|_{x=f^{-1}(z)})|^{-1}p_X(f^{-1}(z)) \quad (79)$$

$$= \int \mathrm{d}x'\delta(f(x')-z)p_X(x')\,, \quad (80)$$

where the second equality follows upon changing variables to $x'' = f(x')$.

We also present an alternative proof the lemma, which relies on:

$$\widehat{U}\,|x\rangle = |\mathrm{Det}(\tfrac{\partial F(x)}{\partial x})|^{1/2}\,|F(x)\rangle\ . \quad (81)$$

The fact that $\widehat{U}\,|x\rangle = c(x)\,|F(x)\rangle$ for some $c(x)$ follows from

$$\widehat{\varphi}\widehat{U}\,|x\rangle = \widehat{U}F(\widehat{\varphi})\,|x\rangle = F(x)\widehat{U}\,|x\rangle\ , \quad (82)$$

and the delta normalization fixes the proportionality factor:

$$\delta(x'-x) = \langle x'|x\rangle = \langle x'|\,\widehat{U}^\dagger\widehat{U}\,|x\rangle \quad (83)$$

$$= \overline{c(x')}c(x)\delta(F(x')-F(x))\,, \quad (84)$$

using that for a $g(x)$ having the unique solution $g(x) = 0$ at x_0, one has

$$\delta(g(x)) = \delta(x-x_0)|\mathrm{Det}(\tfrac{\partial g(x)}{\partial x}|_{x=x_0})|^{-1}\,. \quad (85)$$

Given equation (81), the lemma is immediate:

$$|\langle\boldsymbol{\varphi}|\,\widehat{U}\,|\boldsymbol{m},C\rangle|^2 = |\,\langle\boldsymbol{m},C|\,\widehat{U}^\dagger\,|\boldsymbol{\varphi}\rangle\,|^2 \quad (86)$$

$$= |\mathrm{Det}(\tfrac{\partial F^{-1}(\boldsymbol{\varphi})}{\partial\boldsymbol{\varphi}})|\cdot|\,\langle\boldsymbol{m},C|F^{-1}(\boldsymbol{\varphi})\rangle\,|^2 \quad (87)$$

$$= |\mathrm{Det}(\tfrac{\partial F(x)}{\partial x}|_{x=F^{-1}(\boldsymbol{\varphi})})|^{-1}\mathcal{GP}(\boldsymbol{m}^1,C^{11})(F^{-1}(\boldsymbol{\varphi}))\,, \quad (88)$$

which coincides with the definition of push forward.

□

D. Experiments

In this section we discuss numerical experiments to show that the semi-classical neural networks introduced in Sec. 7 performs on par with its classical counterpart for a simple irregular time series classification task.

D.1. Datasets

Similarly to (Li & Marlin, 2015), to perform controllable experiments with irregular data, we select the UCR time series dataset (Chen et al., 2015). We assume that the input time series is uniformly sampled in $[0, 1]$ and then sample randomly a fraction f of the input time series, where f varies from 1 (no subsampling) to .1 (only 10% of the data is retained). This gives us coordinates x_i and values y_i that can be used to prepare an input Gaussian state as discussed below in the training algorithm section. We select the same random subsampling mask for the the training data and a different one for the test data, which we expect is a good model of a realistic irregular measurement and a more challenging machine learning task.

Since our algorithm has cubic complexity in number of points of the input signal and the purpose of the experiments is only to validate that the semiclassical architecture can perform on par with a classical probabilistic numeric baseline, we restrict the focus here only to the following dataset which has small number of data points: SyntheticControl (train: 300, test: 300, classes: 6, length time series 60).

D.2. Training Algorithm

The training algorithm is composed of two steps. First, we train the GP using Alg. 1. Then we use the resulting posterior mean and covariance – more precisely the posterior mean μ', the cholesky decomposition of the posterior covariance L and its inverse M as inputs to the PNN and SPNN. The PNN discards M, which is used for the momenta sector, which is decoupled for PNN and does not intervene in the prediction. Then we use μ', L, M to sample from a normal distribution that is defined as in Alg. 2, i.e. we set all the other means to zero and the covariance is $k'^{-1} = MM^T$ for the momenta sector and channel 0 while identity elsewhere. This returns a batch of samples $\boldsymbol{z}^i$ that are then passed through the neural network as in Alg. 3 to finally be averaged to produce the logits used for classification. The next section describes the architectures in more details.

Note that GP inference is the bottleneck here and we could use the methods of (Li & Marlin, 2016) to improve on that. Our interest is however not to have a state of the art classifier,

Algorithm 1. PretrainGP

Input: data $\{x_i^b, y_i^b\}$ $b = 1, \dots, B$ (B being data size), grid $\mathcal{X}$, GP kernel parameters Θ.
Train GP: Compute Θ by maximizing marginal likelihood of data in mini-batches:

$$\max_{\Theta} \frac{1}{B} \sum_{b=1}^{B} \log\big(p_{GP}(\boldsymbol{y}^b|\boldsymbol{x}^b, \Theta)\big) \tag{89}$$

Test GP: Compute $\mu_x'^b$, $k_{x,x'}'^b$ from Eq. (1) (main text) for $x, x' \in \mathcal{X}$.
Cholesky: Compute $L^b = \text{cholesky}(k'^b)$, $M^b = \text{cholesky}((k'^b)^{-1})$
Output: $\{\mu'^b\}$, $\{L^b\}$, $\{M^b\}$

Algorithm 2. Sample

Input:μ', L, M (output of Alg. 1), number of channels N_C, number of sample N_{samples}.
Create Normal: $\mathcal{N}(\boldsymbol{m}, C)$ with : $N_c \times |\mathcal{X}| \times 2$ and C: $(N_c \times |\mathcal{X}| \times 2) \times (N_c \times |\mathcal{X}| \times 2)$,

$$m_{x,c,j} = \begin{cases} \mu'_x & c = j = 0 \\ 0 & \text{else} \end{cases}$$

$$C_{x,c,j;x',c',j'} = \begin{cases} k'_{x,x'} & j = j' = c = c' = 0 \\ (k'^{-1})_{x,x'} & j = j' = 1, c{=}c'{=}0 \\ \delta_{x,x'}\delta_{c,c'}\delta_{j,j'} & \text{else} \end{cases}$$

with $k' = LL^T$, $k'^{-1} = MM^T$:
Sample: $\boldsymbol{z}^i$ from $\mathcal{N}(\boldsymbol{m}, C)$ using the reparametrization trick for $i = 1, \dots, N_{\text{samples}}$
Output: $\{\boldsymbol{z}^i\}$

Algorithm 3. Semi-ClassicalNN

Input:μ', L, M (output of Alg. 1), number of channels N_C, β, number of sample N_{samples}, learnable parameters $h^{(\ell)}, b^{(\ell)}$
Sample: $\boldsymbol{z}^i = Sample(\mu', L, M, N_C, N_{\text{samples}})$ (see Alg. 2).
for $\ell = 0$ **to** $L - 1$ **do**
  $\boldsymbol{z}^i = \text{Linear}(\boldsymbol{z}^i, h^{(\ell)}, b^{(\ell)})$
  $\boldsymbol{z}^i = \text{SympecticSofplus}(\boldsymbol{z}^i, \beta)$
end for
$\boldsymbol{z}^i = \text{Linear}(\boldsymbol{z}^i, h^{(L)}, b^{(L)})$
$l_c = N_{\text{samples}}^{-1} \sum_{i=1}^{N_{\text{samples}}} z^i_{x=0,c,j=0}$
Output: Logits l_c

so we test the model in the simplest setting of $O(N^3)$ exact GP inference and simply validate that the performance of the semi-classical neural network architecture is on par with a classical counterpart. Note that separating the GP training from the neural network training allows us to amortize the GP inference as a preprocessing step and have an $O(N^2)$ algorithm for network propagation as in BNN case.

D.3. Architecture

For the GP to interpolate the data we take Matern kernel with $\nu = 0.5$ as implemented in gpytorch (Gardner et al., 2018). We compare three models: 1) a baseline MLP (BNN), 2) a probabilistic numeric MLP (PNN) and 3) a semiclassical version of the PNN (SPNN). All models share the same basic architecture, which is a MLP with three layers. In all cases the hidden sizes are equal to the number of points in the original grid of the data, $|\mathcal{X}|$, times the number of channels N_C. For BNN we parameterize the weight matrices directly i.e. the learnable parameters are the weight matrices. Instead for PNN and SPNN we parametrize the logarithm of the weight matrix as the learnable parameters. We did check that for PNN this did not impact perform comparing to the case where we use the same parameterization of BNN. We do this to ensure simply that the weight matrices of the SPNN are symplectic. Indeed recalling the definitions from the main text, see Sec. 3.2, a linear layer with input/output dimensions $2N$ has learnable parameters $A, B, C \in \mathbb{R}^{N\times N}$ and the weight matrix S is then constructed as:

$$S = \exp\begin{pmatrix} A & \frac{1}{2}(B + B^T) \\ \frac{1}{2}(C + C^T) & -A^T \end{pmatrix}. \tag{90}$$

A, B, C are denoted h in Alg. 3. For PNN we simply take $B = C = 0$ and discard the lower right block. We have also tried to restrict S to be a free matrix and use the parametrization from (de Gosson, 2006) but found that the matrix exponential (which allow use to parametrize more general symplectic matrices) worked best. We used pytorch (Paszke et al., 2019) to run the experiments and initialized all the matrices W using `nn.init.kaiming_uniform_`, with mode fanin and nonlinearity 'relu'. For SPNN we further multiplied the learnable parameters by a scale factor 0.1 which we found important with the training settings described below to prevent large values at the beginning of training.

Finally, we took for non-linearity the softplus for BNN and PNN and its symplectic version of section 7.2 for SPNN. In all cases, $\beta = 0.1$ was chosen. This ensures that in the SPNN we do not create very large values for large negative φ that would make training unstable, and was chosen the same for all models for comparison. We checked that one could get similar results for BNN and PNN by using the more conventional value of $\beta = 1$.

SAMPLING	BNN	PNN	SPNN
1	94	94	95
0.9	84.66 ± 0.47	93.33 ± 1.69	93.00 ± 2.82
0.8	80.66 ± 3.68	91.00 ± 2.82	91.33 ± 2.05
0.7	77.33 ± 6.02	89.00 ± 2.16	86.66 ± 1.25
0.6	74.99 ± 7.87	82.33 ± 5.31	82.00 ± 2.94
0.5	68.33 ± 6.85	75.99 ± 8.49	74.33 ± 11.12
0.4	68.33 ± 5.44	72.66 ± 3.68	72.33 ± 4.11
0.3	66.33 ± 5.25	66.99 ± 4.55	66.66 ± 5.44
0.2	38.99 ± 8.04	42.33 ± 5.25	41.66 ± 4.78
0.1	28.33 ± 8.18	36.00 ± 5.35	40.33 ± 8.05

Table 1. Classification accuracies for a baseline MLP (BNN), the classical probabilistic numeric network (PNN) and the semiclassical network (SPNN). Sampling stands for the subsampling fraction f of the input as explained in the main text. The results are obtained by computing statistics over three random subsampling masks.

D.4. Results

We present results in table 1. We pretrained the GPs using Adam optimizer with learning rate 0.1 from (Paszke et al., 2019) for 20 epochs using batch size of 50. We trained the neural network using default Adam optimizer for 1000 epochs with batch size 50 as well. We used $N_{\text{samples}} = 100$ and $N_C = 2$.

The BNN baseline as well as PNN and SPNN perform on the original unsampled dataset similarly to the MLP used in (Wang et al., 2016) (95 for SyntheticControl), which also has three layers but additionally dropout and uses ReLU instead of softplus. This validates the choice of architecture we made.

The results presented are on the average over three random seeds for the random sampling. The error bars are considerable in all cases due to the fact that different sampling masks lead to different data points that are retained and this choice affects the classification of the signal. When we start to subsample the data (using the same procedure for all models), both the PNN and SPNN n average perform better than the BNN, confirming the findings of (Li & Marlin, 2015; 2016; Finzi et al., 2020) showing the the usefulness of the GP model for irregularly sampled data. (Li & Marlin, 2015) report accuracies averaged over all the 43 UCR datasets and not directly comparable with our restricted setting, while (Li & Marlin, 2016; Finzi et al., 2020) use different datasets for which scalable methods for GP inference are required. Finally, we note that the performance of PNN and SPNN have a considerable overlap within the confidence region of the results, validating the claim that they perform similarly.

E. Details of quantum optical implementation

E.1. State preparation strategies

The state preparation step encodes the input signal $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ into quantum registers. There a several options for that. First we note that while the prescription of lemma 4.1, i.e. to create a prior state and to project onto y_i the registers associated to locations x_i, is appealing conceptually and allows us to formulate inference entirely in quantum language, it is not straightforward to implement. This is because projections are enacted by partial measurements and those give outcome y_i only with a small probability of success. To overcome this, we consider the following alternatives. The simplest is the one explained in the main text, that we repeat here. We use classical hardware to compute the GP posterior as in (1) for a set of points $x \in \mathcal{X}$. Then we create a an input Gaussian state by acting on the vacuum state $|\Omega\rangle$ defined in Sec. 5 with the linear layer $\widehat{D}(\boldsymbol{\xi} = (\boldsymbol{\mu}', 0))\widehat{\omega}(S = A \oplus (A)^{-1})$, where A is a square root of k'. The vacuum state can be created using lasers (Nielsen & Chuang, 2000) and we defer to the next section the implementation of the linear layer. While this procedure is straightforward it incurs a complexity similar to the classical GP inference, that is $O(N^3)$. Another alternative is to use the fact that on quantum hardware we can perform matrix inversion exponentially faster (Harrow et al., 2009). Under some conditions (in particular access to QRAM) one can compute posterior mean and covariance in polylog time (Zhao et al., 2019; Das et al., 2018a). Under similar conditions, quantum singular value decomposition can compute A in polylog time (Kerenidis & Prakash, 2016). The catch in our setting however is that to read out the values of $\boldsymbol{\mu}'$ and A we would still need $O(N^2)$ operations. We leave developing a more efficient quantum implementation of quantum GP inference as in interesting future challenge.

E.2. Linear layer

The implementation of a unitary $\widehat{\omega}(S)$ on a quantum optical computer is a well studied problem and we limit here to give a high level description, see (Weedbrook et al., 2012) for more details. We first decompose the unitary in terms of elementary linear optical gates, we use the group homomorphism property $\widehat{\omega}(S)\widehat{\omega}(S') = \widehat{\omega}(SS')$ together with the Bloch-Messiah decomposition $S = K\Sigma L$ with K, L symplectic and orthogonal and $\Sigma = \text{diag}(\mathrm{e}^{r_1}, \dots, \mathrm{e}^{r_M}, \mathrm{e}^{-r_1}, \dots, \mathrm{e}^{-r_M})$. Σ can be implemented directly using optical parametric amplifiers. The orthogonal matrices K, L can be further decomposed using Givens rotations as product of rotations of two components and implemented in terms of beamsplitters and phase shifters. Note that the number of gates required to decompose an arbitrary matrix can grow quadratically with the dimension (Reck et al., 1994). The bias $\widehat{D}$ can be also easily

implemented (Weedbrook et al., 2012).

E.3. Non linearity

Quantum computers can perform arbitrary computations if given a set of universal gates. For quantum optical computers, one can take the quadratic Hamiltonians and the cubic gate, whose Hamiltonian is $\widehat{\varphi}^3$ (Lloyd & Braunstein, 1999). We can implement a non-linearity with Hamiltonian (30) by approximating it with the truncation of the Taylor series of the function f to order k:

$$\widehat{H} = \sum_{\ell=0}^{k} f_\ell \widehat{H}_\ell\,, \quad \widehat{H}_\ell = \frac{1}{2}\left(\widehat{\pi}\widehat{\varphi}^\ell + \widehat{\varphi}^\ell\widehat{\pi}\right)\,. \tag{91}$$

where for notational simplicity in this section we are going to omit the indices x, a from the quantum fields. We can then use the standard procedure for quantum simulation, see e.g. (Nielsen & Chuang, 2000). A first step is to trade $\mathrm{e}^{im\epsilon\widehat{H}}$ for the m times application of $\prod_{\ell=0}^{k}\mathrm{e}^{i\epsilon f_\ell\widehat{H}_\ell}$, so that the problem boils down on how to implement $\mathrm{e}^{i\epsilon f_\ell\widehat{H}_\ell}$. This is explained in the next proposition:

Proposition E.1. *Denoted $\alpha_\ell = -1/(4(\ell+2))$, define the unitaries*

$$\widehat{U}_{\ell,\epsilon} = \exp\Big(i\epsilon\widehat{H}_\ell\Big) \tag{92}$$

$$\widehat{W}_{\ell,\epsilon} = \exp\big(i\epsilon\alpha_\ell\widehat{\pi}^2\big)\,, \tag{93}$$

$$\widehat{V}_{\ell,\epsilon} = \exp\big(-i\epsilon\widehat{\varphi}^3\big)\widehat{U}^\dagger_{\ell,\epsilon}\exp\big(i\epsilon\widehat{\varphi}^3\big)\widehat{U}_{\ell,\epsilon}\,. \tag{94}$$

We have

$$\widehat{U}_{\ell+1,\epsilon} = \widehat{W}_{\ell,\sqrt{\epsilon}}\widehat{V}_{\ell,\epsilon^{1/4}}\widehat{W}^\dagger_{\ell,\sqrt{\epsilon}}\widehat{V}^\dagger_{\ell,\epsilon^{1/4}} + O(\epsilon^{3/4})\,. \tag{95}$$

Proof. Consider the Trotter formula:

$$\begin{aligned}&[\mathrm{e}^{i\epsilon\widehat{H}_1}\mathrm{e}^{i\epsilon\widehat{H}_2}\cdots\mathrm{e}^{i\epsilon\widehat{H}_k}][\mathrm{e}^{i\epsilon\widehat{H}_k}\mathrm{e}^{i\epsilon\widehat{H}_{k-1}}\cdots\mathrm{e}^{i\epsilon\widehat{H}_1}]\\ &= \mathrm{e}^{2i\epsilon\widehat{H}} + O(\epsilon^3)\,.\end{aligned}$$

The error in replacing $\mathrm{e}^{i\widehat{H}}$ with the m-th power of the l.h.s. is smaller than $\alpha m\epsilon^3$ for some constant α (see Eq. 4.107 of (Nielsen & Chuang, 2000)). So we can concentrate on implementing $\mathrm{e}^{i\epsilon H_\ell}$. Suppose that we know how to implement $\mathrm{e}^{i\epsilon H_\ell}$. Then we show how to construct $\mathrm{e}^{i\epsilon H_{\ell+1}}$ using the universal cubic and linear gates. Note that

$$\begin{aligned}&[\widehat{\varphi}^3, \widehat{H_\ell}] = f_\ell[\widehat{\varphi}^3, \widehat{\pi}\widehat{\varphi}^\ell] = f_\ell\widehat{\varphi}^\ell[\widehat{\varphi}^3, \widehat{\pi}] = 2if_\ell\widehat{\varphi}^\ell\,,\\ &[\widehat{\pi}^2, [\widehat{\varphi}^3, \widehat{H}_\ell]] = 2if_\ell[\widehat{\pi}^2, \widehat{\varphi}^{\ell+2}]\\ &= 2f_\ell(\ell+2)(\widehat{\pi}\widehat{\varphi}^{\ell+1} + \widehat{\varphi}^{\ell+1}\widehat{\pi})\\ &= 4\frac{f_\ell}{f_{\ell+1}}(\ell+2)\widehat{H}_{\ell+1}\,.\end{aligned}$$

Now if we know how to implement two gate $\mathrm{e}^{i\epsilon A}, \mathrm{e}^{i\epsilon B}$, we also know how to implement the gate with Hamiltonian given by their commutator:

$$\begin{aligned}&\mathrm{e}^{-i\sqrt{\epsilon}\widehat{A}}\mathrm{e}^{-i\sqrt{\epsilon}\widehat{B}}\mathrm{e}^{+i\sqrt{\epsilon}\widehat{A}}\mathrm{e}^{+i\sqrt{\epsilon}\widehat{B}}\\ &= \mathrm{e}^{-i\sqrt{\epsilon}(\widehat{A}+\widehat{B})-\frac{1}{2}\epsilon[\widehat{A},\widehat{B}]}\mathrm{e}^{+i\sqrt{\epsilon}(\widehat{A}+\widehat{B})-\frac{1}{2}\epsilon[\widehat{A},\widehat{B}]} + O(\epsilon^{3/2})\\ &= \mathrm{e}^{-\epsilon[\widehat{A},\widehat{B}]} + O(\epsilon^{3/2})\,.\end{aligned}$$

So if we know how to implement $\mathrm{e}^{i\epsilon H_\ell}$, we can implement $\mathrm{e}^{i\epsilon H_{\ell+1}}$ using the fact that $\widehat{\pi}^2$ and $\widehat{\varphi}^3$ are universal gates and using the above formula two times. Since we know how to implement $H_{\ell=0} = \widehat{\pi}$, we can implement all the higher Hamiltonians recursively. □

The number $N_{\widehat{H}}$ of universal gates to implement a gate with Hamiltonian $\widehat{H}$ satisfies the recursion:

$$N_{\widehat{H}_{\ell+1}} = 2N_{\widehat{\pi}^2} + 2(2N_{\widehat{\varphi}^3} + 2N_{\widehat{H}_\ell})\,, \tag{96}$$

where setting $N_{\widehat{\pi}^2} = N_{\widehat{\varphi}^2} = N_{\widehat{H}_0} = 1$ we get the recursion $N_{\widehat{H}_{\ell+1}} = 6 + 4N_{\widehat{H}_\ell}, N_{\widehat{H}_0} = 1$ whose solution is exponential in ℓ, and therefore in k, the truncation parameter of the non-linearity. Depending on the hardware, a low value of k might be required with the proposed procedure.

To understand what function the approximated gates implement, let us consider:

$$\begin{aligned}&h_{\ell,\alpha}(\widehat{\varphi}) := \mathrm{e}^{i\frac{\alpha}{2}(\widehat{\pi}\widehat{\varphi}^\ell+\widehat{\varphi}^\ell\widehat{\pi})}\widehat{\varphi}\mathrm{e}^{-i\frac{\alpha}{2}(\widehat{\pi}\widehat{\varphi}^\ell+\widehat{\varphi}^\ell\widehat{\pi})}\\ &= \widehat{\varphi} + \alpha\widehat{\varphi}^\ell + \frac{\alpha^2}{2}\ell\widehat{\varphi}^{\ell-1}\widehat{\varphi}^\ell + \frac{\alpha^3}{3!}\ell(2\ell-1)\widehat{\varphi}^{2\ell-2}\widehat{\varphi}^\ell + \dots\\ &= \widehat{\varphi}\sum_{j\geq 0}\frac{\alpha^j}{j!}\ell(2\ell-1)(3\ell-2)\dots((j-1)\ell-(j-2))\widehat{\varphi}^{j(\ell-1)}\,,\\ &h_{1,\alpha}(\widehat{\varphi}) = \mathrm{e}^\alpha\widehat{\varphi}\\ &h_{2,\alpha}(\widehat{\varphi}) = \frac{\widehat{\varphi}}{1-\alpha\widehat{\varphi}}\\ &h_{3,\alpha}(\widehat{\varphi}) = \frac{\widehat{\varphi}}{\sqrt{1-2\alpha\widehat{\varphi}^2}}\\ &\dots\,.\end{aligned}$$

So consider the Hamiltonian of softplus by setting

$$\alpha_\ell = \epsilon\frac{(-1)^\ell}{\ell!} \tag{97}$$

and truncating to order k and using Trotter, the function implemented by our procedure is:

$$[\mathrm{e}^{i\epsilon H_1}\cdots\mathrm{e}^{i\epsilon H_k}][\mathrm{e}^{i\epsilon H_k}\cdots\mathrm{e}^{i\epsilon H_1}]\cdot \tag{98}$$

$$\widehat{\varphi}\cdot[\mathrm{e}^{-i\epsilon H_1}\cdots\mathrm{e}^{-i\epsilon H_k}][\mathrm{e}^{-i\epsilon H_k}\cdots\mathrm{e}^{-i\epsilon H_1}] \tag{99}$$

$$= h_{1,\alpha_1}\circ\cdots\circ h_{k,\alpha_k}\circ h_{k,\alpha_k}\circ\cdots\circ h_{1,\alpha_1}(\widehat{\varphi})\,. \tag{100}$$

Define the m-th power of this function by $\sigma_{k,\epsilon,m}(\widehat{\varphi})$.

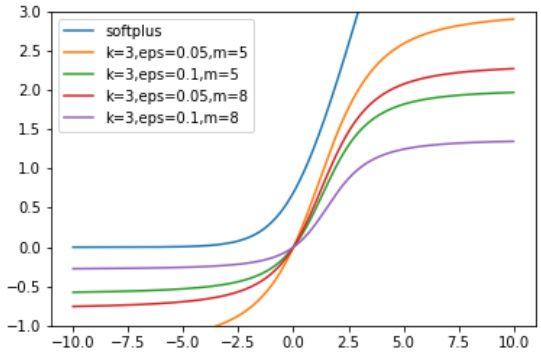

Figure 4. Truncated non-linearity $\sigma_{k,\epsilon,m}(x)$

Figure 4 shows these non-linearity against the original softplus function for different values of the parameters showing the effect of the choice of m and ϵ for $k = 3$.