Tensor Programs I: Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes

Greg Yang[*]
Microsoft Research AI
gregyang@microsoft.com

Abstract

Wide neural networks with random weights and biases are Gaussian processes, as originally observed by Neal (1995) and more recently by Lee et al. (2018) and Matthews et al. (2018) for deep fully-connected networks, as well as by Novak et al. (2019) and Garriga-Alonso et al. (2019) for deep convolutional networks. We show that this Neural Network-Gaussian Process correspondence surprisingly extends to all modern feedforward or recurrent neural networks composed of multilayer perceptron, RNNs (e.g. LSTMs, GRUs), (nD or graph) convolution, pooling, skip connection, attention, batch normalization, and/or layer normalization. More generally, we introduce a language for expressing neural network computations, and our result encompasses all such expressible neural networks. This work serves as a tutorial on the *tensor programs* technique formulated in Yang (2019) and elucidates the Gaussian Process results obtained there. We provide open-source implementations of the Gaussian Process kernels of simple RNN, GRU, transformer, and batchnorm+ReLU network at `github.com/thegregyang/GP4A`.

1 Introduction

Motivated to understand the Bayesian prior in neural networks (NNs), Neal [41] theoretically showed that infinitely wide, shallow neural networks with random weights and biases are Gaussian processes (GPs). He empirically explored this phenomenon over deep networks as well, but this was not proven rigorously until recently [37, 40, 43, 18], with concrete progress made over the intervening years [56, 34, 22, 13]. This neural network-Gaussian process correspondence (NN-GP correspondence) has not only allowed one to transform the *implicit prior* of NNs into *explicit priors* that can be understood analytically [46, 49, 63, 59, 65], but has also created new state-of-the-art kernels by converting from deep neural networks [37, 43]. Yet, so far the focus has dwelled entirely on multilayer perceptrons (MLPs) or simple convolutional neural networks (CNNs). As new architectures are created with blistering speed, a question starts to emerge and reverberate:

Do all infinitely wide, randomly initialized neural networks correspond to Gaussian processes?

Even if the answer is yes, at the current rate where each new architecture warrants its own NN-GP correspondence paper, theory will never catch up to practice. On a more basic level, what does this question even mean for recurrent neural networks?

Our Contributions In this paper, we formulate the notion of a Gaussian process with variable-dimensional output (see Definition 2.1), and show that feedforward and recurrent neural networks

[*]Version 2 update: Introduced Self-Parametrized NETSOR$^+$ programs to handle common NETSOR$^+$ use-cases more naturally (see Appendix C.1); wrote down formal specification of the main languages introduced in this paper (see Appendix J).

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

of *standard architectures* converge to Gaussian processes in this sense as their widths or number of channels go to infinity, when their weights and biases are randomized. *By* ***standard architecture*** *we mean any architecture that is some composition of multilayer perceptrons (MLPs), recurrent neural networks (RNNs) (e.g., Long-Short Term Memory (LSTM) [26] or Gated Recurrent Unit (GRU) [10]), skip connections [24, 27], convolutions [16, 17, 47, 35, 36] or graph convolutions [8, 25, 15, 38, 14, 31], pooling [35, 36], batch normalization (batchnorm) [28], layer normalization [1] and/or attention [2, 55].* Even more broadly, we design a new language, NETSOR, for expressing neural network computations, and show the GP convergence for all such expressible networks. By demonstrating that NETSOR can implement any network of standard architectures, we obtain the aforementioned results as a corollary. The results for RNNs, batchnorm, layernorm, attention, and their combination with other layers are new. We open-source reference implementations[2] for the GP kernels of simple RNN, GRU, transformer, and feedforward batchnorm network; see Fig. 3 for an illustration.

Relation of This Paper with [60] This paper serves several purposes. 1) Introduce the reader to the *tensor programs* technique formulated in [60], using the Neural Network-Gaussian Process Correspondence as motivation. 2) Promote a redesigned set of notations for *tensor programs* that hopefully makes the understanding and the application of this technique easier. 3) Prove a more general version of the Gaussian Process results first presented in [60]. 4) Provide example calculations and reference implementations[2] of the GP kernels for several architectures like the vanilla RNN, GRU, batchnorm network, and transformers.

We assume the reader has not read [60] and seek to explain all results in elementary terms. However, we will provide commentary in footnotes throughout the paper on differences from [60].

Regarding 1), this paper will be the first in a series to explain the *tensor programs* technique, each covering a more powerful type of tensor programs, and each motivated by specific theorems that can be proved or calculations made possible by these new tensor programs. In particular, here we will only talk about tensor programs without matrix transposes. Regarding 3), the results presented here will supersede all results in [60] concerning Gaussian Processes, with one caveat that here we will not cover architectures using both a weight W and its transpose $W^\top$ in its forward pass (but this result will come for free in a later paper in this series).

2 Gaussian Process with Variable-Dimensional Output

We first clarify the notion of a Gaussian process with variable dimension output.

Definition 2.1 (Gaussian Process)**.** We say a random function $f : X \to \mathbb{R}^m$ (with fixed dimensional output) is a Gaussian process if for any finite subset $\{x^1, \ldots, x^k\} \subseteq X$, the random vector $(f(x^1), \ldots, f(x^k)) \in \mathbb{R}^{m \times k}$ is distributed as a km-dimensional Gaussian. If f has variable dimensional output (e.g. f is an RNN), such as when $f(x) \in \mathbb{R}^{l(x)}$ for some length function $l : X \to \mathbb{N}$ [3], then we say f is a Gaussian process if for any finite subset $\{x^1, \ldots, x^k\} \subseteq X$, the random vector $(f(x^1), \ldots, f(x^k))$ is distributed as a $(\sum_i l(x^i))$-dimensional Gaussian.

To illustrate a GP with variable-dimensional output, consider a simple RNN that runs on two input sequences given by the GloVe embeddings [44] [4] of the words of the two sentences

sentence 1 (7 words): "The brown fox jumps over the dog."
sentence 2 (9 words): "The quick brown fox jumps over the lazy dog." $(\star)$

A pseudocode is given in Program 2 in Section 4 (ignore the type annotations like $\mathsf{G}(n), \mathsf{H}(n), \mathsf{A}(n)$ for now). The RNN emits a single scalar after reading each token (in Program 2, this is $v^\top s^{ia}/\sqrt{n}$, where s^{ia} is the RNN state after reading the ith token of the ath sentence, and v is the readout layer); this number takes into account all of the word embeddings read so far. Thus, it will output a total of 7 scalars after reading sentence 1, and a total of 9 scalars after reading sentence 2. To say that this RNN is a GP would imply that all $7 + 9 = 16$ scalars are jointly Gaussian-distributed (corresponding to a

[2] github.com/thegregyang/GP4A

[3] i.e. $f : \prod_{x \in X} \mathbb{R}^{l(x)}$ is a dependent function

[4] The embedding associates each word to a real vector of 100 dimensions such that semantically similar words are mapped to closer vectors

16×16 kernel), over the randomness of the weights and biases imbued during initialization. This is indeed the empirical phenomenon with a width-1000 RNN, and Fig. 2(E) visualizes the the joint distribution of the last scalars output by the RNN at the end of each sentence. It clearly exhibits a Gaussian nature, and perfectly fits the theoretically predicted Gaussian distribution (dashed ovals), which we shall describe in Corollary 5.5.

3 Recap: GP Behavior of a Multilayer Perceptron (MLP)

Before explaining our main results, we first review the argument from prior works [37, 40, 43] for the GP convergence of a wide MLP with randomly initialized weights and biases, and we also demonstrate why such an argument is inadequate for RNNs. Consider an MLP with widths $\{n^l\}_l$, weight matrices $\{W^l \in \mathbb{R}^{n^l \times n^{l-1}}\}_l$, and biases $\{b^l \in \mathbb{R}^{n^l}\}_l$, where l ranges among the layer numbers of the MLP. Its computation is given recursively as

$$h^1(x) = W^1 x + b^1 \qquad \text{and} \qquad h^l(x) = W^l \phi(h^{l-1}(x)) + b^l \text{ for } l \geq 2. \tag{1}$$

At initialization time, suppose $W^l_{\alpha\beta} \sim \mathcal{N}(0, \sigma_w^2/n^{l-1})$ for each $\alpha \in [n^l], \beta \in [n^{l-1}]$, and $b^l_\alpha \sim \mathcal{N}(0, \sigma_b^2)$. Consider two inputs x, x'. Conditioned on $h^{l-1}(x)$ and $h^{l-1}(x')$, iid for each α, $(h^l(x)_\alpha, h^l(x')_\alpha)$ is distributed as

$$\mathcal{N}\left(0, \frac{\sigma_w^2}{n^{l-1}} \begin{pmatrix} \|\phi(h^{l-1}(x))\|^2 & \phi(h^{l-1}(x)) \cdot \phi(h^{l-1}(x')) \\ \phi(h^{l-1}(x)) \cdot \phi(h^{l-1}(x')) & \|\phi(h^{l-1}(x'))\|^2 \end{pmatrix} + \sigma_b^2.\right)$$

If $(h^{l-1}(x)_\alpha, h^{l-1}(x')_\alpha)$ is distributed as $\mathcal{N}(0, \Sigma^{l-1})$, iid for each α, then by a law of large number argument, the covariance matrix above converges to a deterministic limit

$$\Sigma^l \overset{\text{def}}{=} \sigma_w^2 \mathop{\mathbb{E}}_{(z,z') \sim \mathcal{N}(0,\Sigma^{l-1})} \begin{pmatrix} \phi(z)^2 & \phi(z)\phi(z') \\ \phi(z)\phi(z') & \phi(z')^2 \end{pmatrix} + \sigma_b^2$$

as the width $n^{l-1} \to \infty$, making $(h^l(x)_\alpha, h^l(x')_\alpha)$ Gaussian distributed as $\mathcal{N}(0, \Sigma^l)$. Iteratively applying this argument for each l yields the result for a deep MLP. A similar logic works for feedforward CNNs.

Unfortunately, this argument breaks down if the weights $\{W^l\}_l$ are tied, i.e. all W^l are equal to a common matrix W, as in the case of an RNN. In this case, when we condition on the preactivations $h^{l-1}(x), h^{l-1}(x')$ of the previous layer, W is no longer conditionally an iid random Gaussian matrix, and all subsequent reasoning breaks down. We can repair this situation for RNNs in an ad hoc way via the Gaussian conditioning technique (Lemma G.7), but we prefer to set our sights wider, and deal with all standard architectures, and more, in one fell swoop. To this end, we develop a framework based on our new NETSOR language.

4 NETSOR : Language for Expressing Neural Network Computation

To show that networks of all standard architectures converge to GPs, we first show that they can be expressed by the following very general NETSOR language (see Programs 1 and 2 for examples)[5], and then show that any computation expressed this way exhibits GP behavior when its dimensions are large.

Definition 4.1. [6] NETSOR *programs* are straight-line programs, where each variable follows one of three types, G, H, or A (such variables are called *G-vars*, *H-vars*, and *A-vars*), and after input variables, new variables can be introduced by one of the rules `MatMul`, `LinComb`, `Nonlin` to be discussed shortly. G and H are *vector types* and A is a *matrix type*; intuitively, G-vars should be thought of as vectors that are asymptotically Gaussian, H-vars are images of G-vars by coordinatewise nonlinearities, and A-vars are random matrices with iid Gaussian entries. Each type is annotated by dimensionality information:

[5] NETSOR is a specific kind of tensor program; for other variants, see Appendix E.

[6] We keep the definition here informal in terms of programming language convention to be accessible to the general machine learning audience. For those with PL background, see Appendix J.

NETSOR program 1 MLP Computation on Network Input x

```
Input: $W^1 x : \mathsf{G}(n^1)$                    ▷ layer 1 embedding of input
Input: $b^1 : \mathsf{G}(n^1)$                      ▷ layer 1 bias
Input: $W^2 : \mathsf{A}(n^2, n^1)$                 ▷ layer 2 weights
Input: $b^2 : \mathsf{G}(n^2)$                      ▷ layer 2 bias
Input: $v : \mathsf{G}(n^2)$                        ▷ readout layer weights
1: $h^1 := W^1 x + b^1 : \mathsf{G}(n^1)$           ▷ layer 1 preactivation; LinComb
2: $x^1 := \phi(h^1) : \mathsf{H}(n^1)$             ▷ layer 1 activation; Nonlin
3: $\tilde{h}^2 := W^2 x^1 : \mathsf{G}(n^2)$       ▷ MatMul
4: $h^2 := \tilde{h}^2 + b^2 : \mathsf{G}(n^2)$     ▷ layer 2 preactivation; LinComb
5: $x^2 := \phi(h^2) : \mathsf{H}(n^2)$             ▷ layer 2 activation; Nonlin
Output: $v^\top x^2 / \sqrt{n^2}$
```

- If x is a (vector) variable of type G (or H) and has dimension n, we write $x : \mathsf{G}(n)$ (or $x : \mathsf{H}(n)$).
- If A is a (matrix) variable of type A and has size $n_1 \times n_2$, we write $A : \mathsf{A}(n_1, n_2)$.

G is a *subtype* of H, so that $x : \mathsf{G}(n)$ implies $x : \mathsf{H}(n)$. A NETSOR program consists of the following three parts.

Input A set of input G- or A-vars.

Body New variables can be introduced and assigned via the following rules (with *intuition in italics*)

`MatMul` if $A : \mathsf{A}(n_1, n_2)$ and $x : \mathsf{H}(n_2)$, we can form a G-var via matrix-vector product:

$$Ax : \mathsf{G}(n_1), \quad \text{“\textit{random iid matrix times a vector is roughly a Gaussian vector.}”}$$[7]

`LinComb` Suppose $x^1, \ldots, x^k : \mathsf{G}(n)$ are G-vars with the same dimension and $a_1, \ldots a_k \in \mathbb{R}$ are constants. Then we can form their linear combination as a G-var:

$$\sum_{i=1}^{n} a_i x^i : \mathsf{G}(n), \quad \text{“\textit{linear combination of Gaussian vectors is Gaussian.}”}$$

`Nonlin` If $x^1, \ldots, x^k : \mathsf{G}(n)$ are G-vars with the same dimension n and $\phi : \mathbb{R}^k \to \mathbb{R}$, then

$$\phi(x^1, \ldots, x^k) : \mathsf{H}(n), \quad \text{“\textit{image of Gaussian vector is not always Gaussian}”}$$

where ϕ acts coordinatewise.

Output For the purpose of this paper[8], the output of a NETSOR program can be any tuple of scalars, $(v^{1\top} y^1/\sqrt{n_1}, \ldots, v^{k\top} y^k/\sqrt{n_k})$, where $v^1 : \mathsf{G}(n_1); \ldots; v^k : \mathsf{G}(n_k)$ are some input G-vars not used elsewhere (and possibly with duplicates $v^i = v^j$), and $y^1 : \mathsf{H}(n_1); \ldots; y^k : \mathsf{H}(n_k)$ are some H-vars (possibly with duplicates $y^i = y^j$).

Examples Program 1 gives an example of a NETSOR program representing an MLP computation. Note that *we account for the input x through its embedding W^1x, not x itself.* This is because 1) our theorems concern the case where all input G-vars are random; in the context of expressing neural network computation, x is a deterministic input, while W^1x is a Gaussian vector when W^1 has iid Gaussian entries; 2) x has a fixed dimension, while we intend all dimensions (like n^1, n^2) in the NETSOR program to tend to infinity, as we'll describe shortly. Similarly, Program 2 expresses in NETSOR the computation of a simple RNN on two separate input sequences; computation on more input sequences follows the same pattern. Note how weight-sharing is easily expressed in NETSOR

[7]Beware: in a later paper (and in [60], tensor program general case), we will introduce matrix transpose as a valid operation, and in that case, Ax can be very far from a Gaussian, and this intuition is no longer correct. Thus, this intuition is more subtle than it might seem at face value.

[8]In general, the output of a tensor program need not be defined, as most of the time we are concerned with how the H-vars produced over the course of the program interact with each other.

NETSOR program 2 Simple RNN Computation on Two Input Sequences

```
    // Embeddings of two inputs sequences
Input: $Ux^{11}, \dots, Ux^{t1} : \mathsf{G}(n)$
Input: $Ux^{12}, \dots, Ux^{r2} : \mathsf{G}(n)$
    // Weight and bias
Input: $W : \mathsf{A}(n, n)$
Input: $b : \mathsf{G}(n)$
    // Readout weights
Input: $v : \mathsf{G}(n)$
    // Computation on sequence 1
    $h^{11} := Ux^{11} + b : \mathsf{G}(n)$
    $s^{11} := \phi(h^{11}) : \mathsf{H}(n)$
    $\tilde{h}^{21} := Ws^{11} : \mathsf{G}(n)$
    $h^{21} := \tilde{h}^{21} + Ux^{21} + b : \mathsf{G}(n)$
    $s^{21} := \phi(h^{21}) : \mathsf{H}(n)$
    $\vdots$
    $\tilde{h}^{t1} := Ws^{t-1,1} : \mathsf{G}(n)$
    $h^{t1} := \tilde{h}^{t1} + Ux^{t1} + b : \mathsf{G}(n)$
    $s^{t1} := \phi(h^{t1}) : \mathsf{H}(n)$
    // Computation on sequence 2
    $h^{12} := Ux^{12} + b : \mathsf{G}(n)$
    $s^{12} := \phi(h^{12}) : \mathsf{H}(n)$
    $\tilde{h}^{22} := Ws^{12} : \mathsf{G}(n)$
    $h^{22} := \tilde{h}^{22} + Ux^{22} + b : \mathsf{G}(n)$
    $s^{22} := \phi(h^{22}) : \mathsf{H}(n)$
    $\vdots$
    $\tilde{h}^{r2} := Ws^{r-1,2} : \mathsf{G}(n)$
    $h^{r2} := \tilde{h}^{r2} + Ux^{r2} + b : \mathsf{G}(n)$
    $s^{r2} := \phi(h^{r2}) : \mathsf{H}(n)$
Output: $(v^\top s^{11}/\sqrt{n}, \dots, v^\top s^{t1}/\sqrt{n},$
        $v^\top s^{12}/\sqrt{n}, \dots, v^\top s^{r2}/\sqrt{n})$
```

because we can re-use A-vars arbitrarily. Appendix A shows more examples of standard architectures in NETSOR and NETSOR$^+$.

More generally, we can allow the nonlinearities in `Nonlin` to depend on parameters; this will be necessary to express layernorm and attention (see Appendix A). We capture this idea in a new rule:

`Nonlin`$^+$ Suppose $x^1, \dots, x^k : \mathsf{G}(n)$ are G-vars with the same dimension n and $\theta_1, \dots, \theta_t \in \mathbb{R}$ possibly depend on G-vars already defined. If $\phi(-; -) : \mathbb{R}^k \times \mathbb{R}^t \to \mathbb{R}$, then

$$\phi(x^1, \dots, x^k; \theta_1, \dots, \theta_t) : \mathsf{H}(n),$$

where ϕ acts coordinatewise.

Definition 4.2. NETSOR$^+$ programs are NETSOR programs allowing `Nonlin`$^+$ rules.

NETSOR and NETSOR$^+$ specify different kinds of *tensor programs*; in Appendix E we discuss other kinds that are semantically equivalent. In a future paper, we shall study the effect of allowing matrix transposes as an operation on A-vars.

Remark 4.3. In this paper, in `Nonlin`$^+$, we will only instantiate θ_j with continuous functions of "empirical moments" of the form $n^{-1} \sum_{i=1}^n \psi(y^1, \dots, y^r)$ for some set of G-vars $\{y_i\}_i$. A key consequence of our scaling limit result is that these "empirical moments" converge almost surely to a deterministic limit under very general conditions (Theorems 5.4 and C.4), so that $\phi(-; \Theta)$ is, under suitable smoothness conditions (Definition C.1), approximately a fixed nonlinearity when n is large. Thus, we should intuitively treat `Nonlin`$^+$ as `Nonlin` but with the nonlinearity determined automatically by the NETSOR program itself.

`Nonlin`$^+$ expands the expressible computation quite broadly, but to keep the main text lean and focused on the key ideas behind tensor programs, we relegate a more thorough discussion of `Nonlin`$^+$ in the appendix (see Appendices C, D and I).

5 Computing the GP Kernel from a NETSOR Encoding of a Neural Network

For readers who wish to be convinced that NETSOR (or NETSOR$^+$) can express standard architectures, see Appendix A. In this section, we show that any architecture expressible in NETSOR and satisfies some mild conditions will exhibit Gaussian Process behavior in the large width limit.

In this section, we make the following simplifying assumption on the dimensions of the program and the randomness of the variables.

Assumption 5.1. *Fix a* NETSOR *program. For simplicity, assume all dimensions in the program are equal to n. Suppose for each A-var $W : \mathsf{A}(n, n)$, we sample $W_{\alpha\beta} \sim \mathcal{N}(0, \sigma_W^2/n)$ for some $\sigma_W^2 > 0$, and for each $\alpha \in [n]$, we sample, i.i.d., $\{x_\alpha : x \text{ is input G-var}\} \sim \mathcal{N}(\mu^{\text{in}}, \Sigma^{\text{in}})$ for some mean μ^{in} and (possibly singular) covariance Σ^{in} over input G-vars.*

The constraint on the dimensions can be removed easily; see Appendix F. This sampling induces randomness in all variables created in the program, and we shall characterize this randomness shortly. We first review some notation that will be used immediately.

Notation In this paper, a *kernel* Σ *on a set* X is a symmetric function $\Sigma : X \times X \to \mathbb{R}$ such that

$$\sum_{i=1}^{m} \sum_{j=1}^{m} c_i c_j \Sigma(x_i, x_j) \geq 0$$

holds for any $m \in \mathbb{N}$, $x_1, \ldots, x_m \in X$, and $c_1, \ldots, c_m \in X$. Given a kernel Σ on a set of G-vars, we will both treat it as matrix and as a function, depending on the context. **Function Notation** As a function, $\Sigma(g, g')$ is the value of Σ on the pair of G-vars (g, g'). If $G = \{g^1, \ldots, g^k\}$ is a set of G-vars, then we also denote by $\Sigma(g, G)$ the row vector $\{\Sigma(g, g^1), \ldots, \Sigma(g, g^k)\}$. Likewise $\Sigma(G, g)$ is the column vector with the same values. If $G' = \{g^{1\prime}, \ldots, g^{l\prime}\}$ is another set of G-vars (possible with overlap with G), then $\Sigma(G, G')$ is the matrix $\{\Sigma(g^i, g^{j\prime}) : i \in [k], j \in [l]\}$. **Restriction Notation** We also use the "restriction" notation $\Sigma|_G$ to denote the square matrix $\Sigma(G, G)$ in a more concise way. **Matrix Notation** When an association of indices to G-vars is clear from context, we will also write Σ_{ij} for the corresponding value of Σ on the pair of ith and jth G-vars. Juxtaposition implies matrix multiplication, e.g. $\Sigma\Omega$ means matrix product if Ω is a matrix of appropriate size. **Indices Notation** We will both use superscripts and subscripts for indices. We will never multiply in subscript or superscript, so juxtaposition of indices like $W^{ib}_{\alpha\beta}$ is the same as $W^{i,b}_{\alpha,\beta}$. **H-vars as Both Symbols and Vectors** An H-var will be considered both as a symbol (like in $\Sigma(g, g')$ above) as well as the corresponding length n vector (like in Theorem 5.4 below), depending on the context.

Definition 5.2. In the setting of Assumption 5.1, we extend μ^{in} and Σ^{in} to μ and Σ that resp. take a single and a pair of G-vars and both output to $\mathbb{R}$. Intuitively, μ specifies the mean coordinate of a G-var, and Σ specifies the coordinatewise covariance of a pair of G-vars; this is formalized in Theorem 5.4 below. Index all the G-vars in the program as $g^1, \ldots, g^M$ (including input G-vars), in the order of appearance in the program. For any pair of G-vars g, g' (among $g^1, \ldots, g^M$), we define recursively

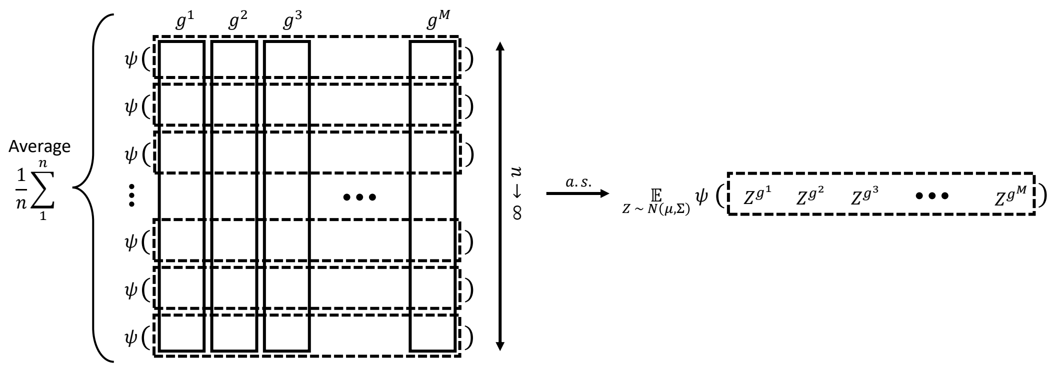
$$\mu(g) = \begin{cases} \mu^{\text{in}}(g) & \text{if } g \text{ is input} \\ \sum_i a_i \mu(y^i) & \text{if } g = \sum_i a_i y^i \text{, introduced by } \texttt{LinComb} \text{ ,} \\ 0 & \text{otherwise} \end{cases}$$

$$\Sigma(g, g') = \begin{cases} \Sigma^{\text{in}}(g, g') & \text{if } g, g' \text{ are inputs} \\ \sum_i a_i \Sigma(y^i, g') & \text{if } g = \sum_i a_i y^i \text{, introduced by } \texttt{LinComb} \\ \sum_i a_i \Sigma(g, y^i) & \text{if } g' = \sum_i a_i y^i \text{, introduced by } \texttt{LinComb} \\ \sigma_W^2 \, \mathbb{E}_Z \, \phi(Z)\bar{\phi}(Z) & \text{if } g = Wh, g' = Wh' \text{, introduced by } \texttt{MatMul} \text{ w/ same A-var } W \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where

- y^i are G-vars for all i
- $(h : \mathsf{H}(n))$ was introduced by the `Nonlin` with $h := \phi(g^1, \ldots, g^M)$, h' was introduced by `Nonlin` with $h' := \bar{\phi}(g^1, \ldots, g^M)$ (where WLOG we have padded the input slots of ϕ and $\bar{\phi}$ to account for all G-vars)
- $Z \sim \mathcal{N}(\mu, \Sigma)$ is a random Gaussian vector with 1 entry for each G-var in the program.

Note that since ϕ and $\bar{\phi}$ only depends on entries of Z corresponding to previous G-vars, the expectation $\mathbb{E}_Z \, \phi(Z)\bar{\phi}(Z)$ only depends on entries of μ and Σ already defined, so there is no circular logic in this recursive definition of μ and Σ. See Appendix B.1.1 for a simple, worked-out example of how to recursively compute μ and Σ for Program 1.



g^1 g^2 g^3 g^M
Average
$\frac{1}{n}\sum_{1}^{n}$
$\psi($ $)$
$n \to \infty$
$a.s.$
$\mathbb{E}_{Z\sim N(\mu,\Sigma)} \psi$
Z^{g^1} Z^{g^2} Z^{g^3} Z^{g^M}

Figure 1: An illustration of the NETSOR Master Theorem Theorem 5.4.

For our main theorems, we isolate a very general class of nonlinearities that we are concerned with.

Definition 5.3. We say a function $\phi : \mathbb{R}^k \to \mathbb{R}$ is *controlled* if $|\phi(x)|$ is bounded by a function of the form $e^{C\|x\|^{2-\epsilon}+c}$ with $C, c, \epsilon > 0$

Controlled functions can explode faster than exponential but are still L^1 and L^2-integrable against Gaussian measures. Additionally, there is no constraint on the smoothness of ϕ here. Thus this definition captures almost all functions we would care about in practice.

The metric structure of the final layer representations of inputs under a deep neural network often reveals semantical information about the inputs. This structure is reflected in the inner products between pairs of such representations, e.g. $s^{t1\top} s^{r2}/n$ for s^{t1} and s^{r2} in Program 2. The following Master Theorem allows one to compute such inner products, and much more, for a wide network at initialization time (take ψ below to be $\psi(z^1, \ldots, z^M) \overset{\text{def}}{=} z^{M-1}z^M$).

Theorem 5.4 (NETSOR Master Theorem)**.** [9] *Fix any* NETSOR *program satisfying Assumption 5.1 and with all nonlinearities controlled. If $g^1, \ldots, g^M$ are all of the G-vars in the entire program, including all input G-vars, then for any controlled $\psi : \mathbb{R}^M \to \mathbb{R}$, as $n \to \infty$,*

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g^1_\alpha, \ldots, g^M_\alpha) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z) = \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z^{g^1}, \ldots, Z^{g^M}),$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence, $Z = (Z^{g^1}, \ldots, Z^{g^M}) \in \mathbb{R}^M$, and $\mu = \{\mu(g^i)\}_{i=1}^M \in \mathbb{R}^M$ and $\Sigma = \{\Sigma(g^i, g^j)\}_{i,j=1}^M \in \mathbb{R}^{M\times M}$ are given in Eq. (2). See Fig. 1 for an illustration.

Intuitively, Theorem 5.4 says, for each α, $(g^1_\alpha, \ldots, g^M_\alpha) \approx \mathcal{N}(\mu, \Sigma)$ in the large n limit, and each α-slice appears to be "iid" from the point of view of the empirical average by any controlled function ψ. The proof of Theorem 5.4 is given in Appendix H.

Combining Theorem 5.4 with Proposition G.4, we can straightforwardly calculate the output distribution of a NETSOR program.

Corollary 5.5 (Computing the GP Kernel)**.** *Adopt the same assumptions and notations as in Theorem 5.4. If the program outputs $(v^\top x^1/\sqrt{n}, \ldots, v^\top x^k/\sqrt{n})$, where*

- *$v : \mathsf{G}(n), v_\alpha \sim \mathcal{N}(0, \sigma_v^2)$, is an input G-var not used elsewhere in the program and is sampled independently from all other G-vars, and*
- *x^i was introduced as $x^i := \phi^i(g^1, \ldots, g^M)$*

then the output vector converges in distribution to $\mathcal{N}(0, K)$ where

$$K_{ij} = \sigma_v^2 \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \phi^i(Z)\phi^j(Z), \quad \textit{with } \mu, \Sigma \textit{ defined in Eq. (2).} \tag{3}$$

[9]Difference with [60, Thm 4.3]: We have gotten rid of the "rank convergence" assumption by showing that it comes for free. See CoreSet and Lemma H.4 in Appendix H.

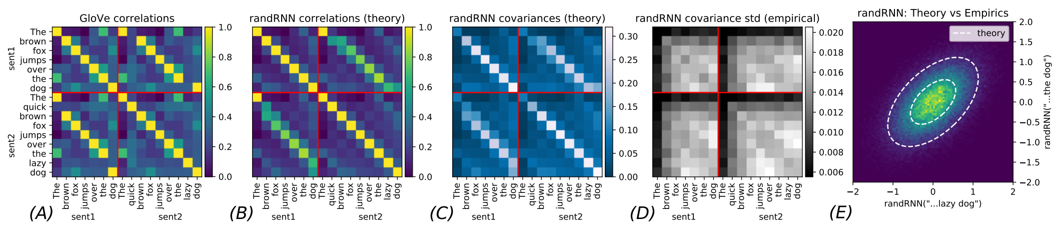


Figure 2: *Infinite-width theory is highly predictive for simple RNN (Program 2) with 1000 neurons and erf activation.* We pass two sentences ("The brown fox jumps over the dog" and "The quick brown fox jumps over the lazy dog") by their word GloVe embeddings into randomly initialized simple RNNs. **(A)** Cosine distances between each pair of word GloVe embeddings. **(B)** Correlation matrix of the limiting Gaussian that Program 2 output vector converges to. Each row/column corresponds to an embedding of of the sentence up to that word. **(C)** *Covariance* matrix of the same. See Appendix B.2 for algorithm to compute this covariance. **(D)** Entrywise standard deviation of empirical covariance around (C), as measured from 100 random simple RNNs. Note the deviations are at least an order of magnitude smaller than the limiting values (C), for 1000 neurons. **(E)** Visualizing the joint distribution of the final outputs of the RNN at the end of each sentence, i.e. $(v^\top s^{t1}/\sqrt{n}, v^\top s^{r2}/\sqrt{n})$ in Program 2. We sampled 100,000 simple RNNs and plotted the 2d histogram as a heatmap. Simultaneously, we plot the limiting Gaussian level curves predicted by our theory, which fit the simulations perfectly.

Intuitively, this corollary follows from the fact that, for any finite n, the output vector is some Gaussian $\mathcal{N}(0, \tilde{K})$ conditioned on $x^1, \ldots, x^k$, and the covariance $\tilde{K}$ converges to a deterministic covariance K, causing the output vector to converge in distribution to $\mathcal{N}(0, K)$ as well. The case when we have multiple distinct v^i (allowed by Definition 4.1) can be obtained easily as well (see Proposition G.4).

Following Corollary 5.5 and its extensions below, the convergence of standard architectures to Gaussian Processes becomes obvious: Express the marginal of the distribution on every finite set of inputs as a NETSOR (or NETSOR$^+$) program, and then apply Corollary 5.5. We summarize the result below.

Corollary 5.6. *If its nonlinearities are controlled (Definition 5.3), then a (possibly recurrent) neural network of standard architecture converges to a Gaussian process in finite-dimensional distribution* [10] *in the sense of Definition 2.1 as its widths go to infinity and each of its weights W and biases b are randomized as $W_{\alpha\beta} \sim \mathcal{N}(0, \sigma_W^2/n), b_\alpha \sim \mathcal{N}(\mu_b, \sigma_b^2)$ for a collection of sampling hyperparameters $\{\sigma_W\}_W, \{\mu_b, \sigma_b\}_b$. If its nonlinearities are more generally parametrized and are parameter-controlled (Definition C.1), such as in the case of attention models or where layernorm is involved, then the same result holds as long as Assumption C.3 also holds.*

An Empirical Demonstration Despite being about infinite width, our theory is highly predictive for finite-width networks, as shown in Fig. 2. As in Section 2, we randomly initialize a simple RNN (Program 2) with 1000 neurons and erf activation (we choose erf instead of tanh because it simplifies kernel calculations; see Appendix B.2 for the derivation of the algorithm to compute the kernel). We pass the two sentences in ($\star$) to the random RNN by their GloVe embeddings. After processing each token, the RNN outputs a scalar, as before, and over the two input sequences, the RNN outputs $7 + 9 = 16$ scalars in total. Our result Corollary 5.5 implies that, as the width of the RNN grows to infinity, these 16 scalars are distributed jointly as a Gaussian. Fig. 2(E) illustrates this is indeed the case for the marginal on 2 scalars, as discussed in Section 2. We also compare our theoretically derived, infinite-width covariance of the 16 scalars (Fig. 2(C)) with the empirical finite-width covariance obtained from multiple random initializations. We find that the empirical covariance, as predicted, concentrates around the theoretical, and the entrywise standard deviation is typically at least an order of magnitude lower than the values themselves (Fig. 2(D)) (with width 1000 RNNs). The random RNN is clearly doing nontrivial context embedding, as seen by comparing the

[10]Stronger convergence results, such as convergence in distribution with respect to some topology on functions on $\mathbb{R}^d$, would be available if one can show additionally the *tightness* of the random neural networks under this topology. However, here we are content with convergence of finite-dimensional marginals of the stochastic processes.

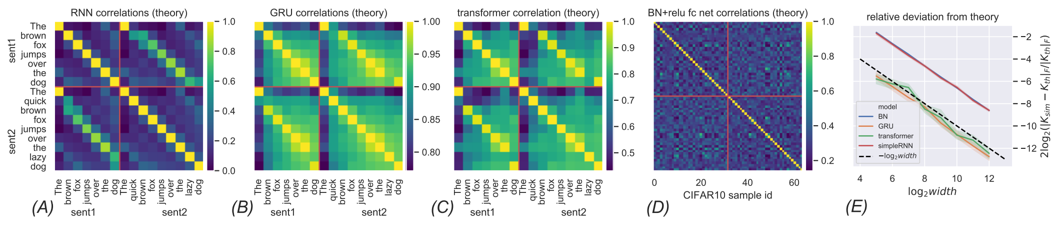


Figure 3: *Infinite-width GP kernels (more precisely, their correlation matrices) for which we provide reference implementations, and the deviation of finite-width simulations from the corresponding infinite-width limits.* **(A) – (C)** The correlation matrices of the GP kernels for the simple RNN (same as in Fig. 2; see Program 2 for the architecture and Appendix B.2 for derivation), GRU (Program 5; Appendix B.5), and transformer (Program 10; Appendix D.3), with input sequences given by the GloVe embeddings of ($\star$). **(D)** The correlation matrix of the GP kernel for a feedforward, fully-connected network with batchnorm+ReLU (batchnorm followed by ReLU) as nonlinearity (see Appendix B.3 for derivation). The inputs are the first 64 CIFAR10 images, split into two batches of 32 each. The red lines indicate the batch split. **(E)** For each architecture above, we independently randomly initialize 100 networks for each width among $[2^5, 2^6, \ldots, 2^{13}]$. We calculate the empirical kernel of the network outputs and plot its (relative) Frobenius distance to the infinite-width kernel. This Frobenius distance drops like $1/\sqrt{width}$ as one would expect from a central limit intuition. See our code[2] for `Python` implementations of these kernels and the code for generating this figure.

correlation matrix of the 16 scalars Fig. 2(B) (context-sensitive) with the matrix of cosine distances (i.e. correlations) between the GloVe embeddings Fig. 2(A) (context-insensitive). A tell-tale sign is the entry corresponding to ("lazy", "dog"): even though as words, they are not semantically similar (so that the entry in Fig. 2(A) is small), the random RNN understands that the two sentences resp. up to "lazy" and "dog" have been very similar (so that the entry in Fig. 2(B) is large). Given the precision of our theoretical predictions, we expect analyses of the equations derived here will lead to many nontrivial insights about recurrent (and other) neural network behavior in practice, which we leave for future work.

Examples and Extensions: A Brief Guide to the Appendix Appendix B contains a plethora of worked-out examples of the kernel computation for different architectures, starting from the known case of MLP to the new results of RNN (as shown in Fig. 2), GRU, batchnorm, and others. At this point, we recommend the reader to follow along some of those examples to solidify the understanding of Theorem 5.4.

A Master Theorem for NETSOR$^+$ can be similarly proved. This is stated in Appendix C and can be proved easily given the proof of Theorem 5.4; see Appendix I. Appendix D works out examples of kernel computations for layernorm and transformer, which can only be expressed through NETSOR$^+$. Fig. 3 illustrates the kernels of simple RNN, GRU, transformer, and a batchnorm+ReLU network, and confirms that the finite width simulations tend to the infinite-width, theoretical kernels.

We also discuss different variants of NETSOR and NETSOR$^+$ in Appendix E which trade off syntactical simplicity with ease of use, but are semantically equivalent to NETSOR or NETSOR$^+$. Appendix F discusses the case when the dimensions of a program need not be equal. With the appropriate setup, a Master Theorem in this case can be proved similarly (Theorem F.4).

6 Related Works

NN-GP Correspondence Many works have observed the neural network-Gaussian process correspondence (NN-GP correspondence) for subsets of standard architectures [56, 34, 22, 13, 37, 40, 43]. Others have exploited this NN-GP correspondence implicitly or explicitly to build new models [11, 33, 12, 57, 58, 7, 54, 32, 4, 6, 18, 43]. In particular, by directly converting NN to GP using this correspondence, Lee et al. [37] constructed the state-of-the-art (SOTA) permutation-invariant GP on MNIST, and Novak et al. [43] was until recently SOTA on CIFAR10 for any GP with untrainable kernel. Additionally, the NN-GP correspondence has led to new understanding of neural network training and generalization [42, 53, 61].

In this paper, we generalized the NN-GP correspondence to *standard architectures* and very general nonlinearities (controlled functions; see Definition 5.3). In contrast, Matthews et al. [40] requires ϕ to be linearly bounded in norm; Daniely et al. [13] requires ϕ be twice-differentiable with $|\phi|, |\phi'|, |\phi''|$ all bounded, or that $\phi = \text{ReLU}$; and a sufficient condition given in Novak et al. [43] is that ϕ' exists and is bounded by $\exp(O(x^{2-\epsilon}))$, though it is unclear how the more general set of 3 conditions given there (in their section E.4) compares with ours.

Signal Propagation in Neural Networks A long line of work starting with Glorot and Bengio [20] and He et al. [23] studies the effect of initialization in deep neural networks [46, 50, 63, 62, 21, 9, 64, 45], for example, what is the best initialization scheme to avoid gradient vanishing? These works apply the same calculations of covariances as we do for calculating Σ here, though in a much more restricted set of architectures, and they are typically more concerned with the dynamics of such covariances with depth.

Reservoir Computing In reservoir computing [30, 39, 51], sequence processing is typically done by a randomly initialized recurrent neural network. A sequence of inputs is fed step by step into the network, and a final readout layer transforms the random RNN's state into an output. The only trainable parameters are the readout layer, but not the random RNN itself. Thus, in the infinite-width limit, reservoir computing corresponds exactly to GP inference with the RNN kernel computed in Appendix B.2.

7 Conclusion

We formulated the notion of Gaussian process with variable-dimensional outputs and showed that randomly initialized, wide feedforward and recurrent neural networks of standard architectures converge in distribution to Gaussian processes in such a sense. This significantly generalizes prior work on the NN-GP correspondence. We did so by introducing NETSOR, a language for expressing computation common in deep learning, including neural networks of standard architecture, along with a theorem (Theorem 5.4) characterizing the behavior of a NETSOR program as its tensors are randomized and their dimensions tend to infinity; many examples and extensions are exhibited in the appendix. Finally, we empirically verified our theory for simple RNN, GRU, transformer, and batchnorm (Fig. 3) and open-sourced implementations of the corresponding infinite-width limit kernels at `github.com/thegregyang/GP4A`. In the next paper in this series, we will introduce a more powerful version of tensor program that allows matrix transposes, and use this tool to compute Neural Tangent Kernel [29] for any architecture.

Acknowlegements

I'm very thankful for my buddy Hadi Salman who is always ready to help and who donated a lot of time helping me write the detailed examples for MLP and RNN kernels. I'd also like to thank Mimee Xu who read the first versions of this paper and provided valuable feedback. Finally, allow me to express my appreciation for myriads of friends and collaborators who have helped me improve this paper in one way or another: Sam Schoenholz, Yihe Dong, Judy Shen, Alessandro Sordoni, Huishuai Zhang, George Phillip, Vinay Rao, Sebastien Bubeck, Zeyuan Allen-Zhu, Kyle Aitkens, Chunyuan Li, Alex Polozov, Ilya Razenshteyn, Jianfeng Gao, Pengchuan Zhang, Jascha Sohl-Dickstein, Jeffrey Pennington, and others.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016. URL http://arxiv.org/abs/1607.06450. 00329 arXiv: 1607.06450.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, September 2014. URL http://arxiv.org/abs/1409.0473. arXiv: 1409.0473.

[3] Mohsen Bayati and Andrea Montanari. The dynamics of message passing on dense graphs, with applications to compressed sensing. *IEEE Transactions on Information Theory*, 57(2): 764–785, February 2011. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.2010.2094817. URL http://arxiv.org/abs/1001.3448. arXiv: 1001.3448.

[4] Kenneth Blomqvist, Samuel Kaski, and Markus Heinonen. Deep convolutional Gaussian processes. *arXiv preprint arXiv:1810.03052*, 2018.

[5] Erwin Bolthausen. An iterative construction of solutions of the TAP equations for the Sherrington-Kirkpatrick model. *arXiv:1201.2891 [cond-mat, physics:math-ph]*, January 2012. URL http://arxiv.org/abs/1201.2891. arXiv: 1201.2891.

[6] Anastasia Borovykh. A gaussian process perspective on convolutional neural networks. *arXiv preprint arXiv:1810.10798*, 2018.

[7] John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*, 2017.

[8] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]*, December 2013.

[9] Minmin Chen, Jeffrey Pennington, and Samuel Schoenholz. Dynamical Isometry and a Mean Field Theory of RNNs: Gating Enables Signal Propagation in Recurrent Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 873–882, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. URL http://proceedings.mlr.press/v80/chen18i.html.

[10] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, June 2014. URL http://arxiv.org/abs/1406.1078. arXiv: 1406.1078.

[11] Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009. URL http://papers.nips.cc/paper/3628-kernel-methods-for-deep-learning.

[12] Andreas Damianou and Neil Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

[13] Amit Daniely, Roy Frostig, and Yoram Singer. Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2253–2261. Curran Associates, Inc., 2016.

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *arXiv:1606.09375 [cs, stat]*, June 2016.

[15] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015.

[16] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975.

[17] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[18] Adrià Garriga-Alonso, Laurence Aitchison, and Carl Edward Rasmussen. Deep Convolutional Networks as shallow Gaussian Processes. *arXiv:1808.05587 [cs, stat]*, August 2018. URL http://arxiv.org/abs/1808.05587. arXiv: 1808.05587.

[19] Alan Genz, Frank Bretz, Tetsuhisa Miwa, Xuefei Mi, Friedrich Leisch, Fabian Scheipl, and Torsten Hothorn. *mvtnorm: Multivariate Normal and t Distributions*, 2019. URL https://CRAN.R-project.org/package=mvtnorm. R package version 1.0-11.

[20] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html. 02641.

[21] Boris Hanin and David Rolnick. How to Start Training: The Effect of Initialization and Architecture. *arXiv:1803.01719 [cs, stat]*, March 2018. URL http://arxiv.org/abs/1803.01719. arXiv: 1803.01719.

[22] Tamir Hazan and Tommi Jaakkola. Steps Toward Deep Kernel Methods from Infinite Neural Networks. *arXiv:1508.05133 [cs]*, August 2015. URL http://arxiv.org/abs/1508.05133. arXiv: 1508.05133.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. URL http://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. pages 770–778, 2016. URL https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

[25] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep Convolutional Networks on Graph-Structured Data. *arXiv:1506.05163 [cs]*, June 2015.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[27] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, August 2016. URL http://arxiv.org/abs/1608.06993. arXiv: 1608.06993.

[28] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, February 2015. URL http://arxiv.org/abs/1502.03167. 04135.

[29] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. *arXiv:1806.07572 [cs, math, stat]*, June 2018. URL http://arxiv.org/abs/1806.07572. 00000 arXiv: 1806.07572.

[30] Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, September 2007. ISSN 1941-6016. doi: 10.4249/scholarpedia.2330.

[31] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*, September 2016.

[32] Vinayak Kumar, Vaibhav Singh, PK Srijith, and Andreas Damianou. Deep Gaussian Processes with Convolutional Kernels. *arXiv preprint arXiv:1806.01655*, 2018.

[33] Neil D Lawrence and Andrew J Moore. Hierarchical Gaussian process latent variable models. In *Proceedings of the 24th international conference on Machine learning*, pages 481–488. ACM, 2007.

[34] Nicolas Le Roux and Yoshua Bengio. Continuous neural networks. In *Artificial Intelligence and Statistics*, pages 404–411, 2007.

[35] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[36] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[37] Jaehoon Lee, Yasaman Bahri, Roman Novak, Sam Schoenholz, Jeffrey Pennington, and Jascha Sohl-dickstein. Deep Neural Networks as Gaussian Processes. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1EA-M-0Z.

[38] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *arXiv:1511.05493 [cs, stat]*, November 2015.

[39] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14(11):2531–2560, November 2002. ISSN 0899-7667, 1530-888X. doi: 10.1162/089976602760407955.

[40] Alexander G. de G. Matthews, Mark Rowland, Jiri Hron, Richard E. Turner, and Zoubin Ghahramani. Gaussian Process Behaviour in Wide Deep Neural Networks. *arXiv:1804.11271 [cs, stat]*, April 2018. URL http://arxiv.org/abs/1804.11271. arXiv: 1804.11271.

[41] Radford M Neal. *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD Thesis, University of Toronto, 1995.

[42] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and Generalization in Neural Networks: an Empirical Study. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJC2SzZCW.

[43] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian Deep Convolutional Networks with Many Channels are Gaussian Processes. *arXiv preprint arXiv:1810.05148*, 2018.

[44] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. pages 1532–1543. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1162. URL http://aclweb.org/anthology/D14-1162. 04219.

[45] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4788–4798. Curran Associates, Inc., 2017. 00004.

[46] Ben Poole, Subhaneil Lahiri, Maithreyi Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances In Neural Information Processing Systems*, pages 3360–3368, 2016. 00047.

[47] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[48] Nico Schlömer. QuadPy: Numerical integration (quadrature, cubature) in Python, 2016–. URL `https://github.com/nschloe/quadpy`. [Online; accessed <today>].

[49] Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep Information Propagation. *arXiv:1611.01232 [cs, stat]*, November 2016. URL `http://arxiv.org/abs/1611.01232`. arXiv: 1611.01232.

[50] Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep Information Propagation. 2017. URL `https://openreview.net/pdf?id=H1W1UN9gg`.

[51] Benjamin Schrauwen. An overview of reservoir computing: Theory, applications and implementations. page 12, 2007.

[52] Elias M. Stein and Rami Shakarchi. *Functional analysis: introduction to further topics in analysis*. Princeton University Press, 2011.

[53] Guillermo Valle-Pérez, Chico Q. Camargo, and Ard A. Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *arXiv:1805.08522 [cs, stat]*, May 2018.

[54] Mark van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional Gaussian Processes. In *Advances in Neural Information Processing Systems 30*, pages 2849–2858, 2017.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, \Lukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[56] Christopher K I Williams. Computing with Infinite Networks. In *Advances in neural information processing systems*, page 7, 1997.

[57] Andrew G Wilson, Zhiting Hu, Ruslan R Salakhutdinov, and Eric P Xing. Stochastic Variational Deep Kernel Learning. In *Advances in Neural Information Processing Systems*, pages 2586–2594, 2016.

[58] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[59] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5393–5402, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. URL `http://proceedings.mlr.press/v80/xiao18a.html`.

[60] Greg Yang. Scaling Limits of Wide Neural Networks with Weight Sharing: Gaussian Process Behavior, Gradient Independence, and Neural Tangent Kernel Derivation. *arXiv:1902.04760 [cond-mat, physics:math-ph, stat]*, February 2019.

[61] Greg Yang and Hadi Salman. A Fine-Grained Spectral Perspective on Neural Networks. July 2019.

[62] Greg Yang and Sam S. Schoenholz. Deep Mean Field Theory: Layerwise Variance and Width Variation as Methods to Control Gradient Explosion. February 2018. URL https://openreview.net/forum?id=rJGY8GbR-.

[63] Greg Yang and Samuel S. Schoenholz. Mean Field Residual Network: On the Edge of Chaos. In *Advances in neural information processing systems*, 2017.

[64] Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A Mean Field Theory of Batch Normalization. September 2018. URL https://openreview.net/forum?id=SyMDXnCcF7.

[65] Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A Mean Field Theory of Batch Normalization. *arXiv:1902.08129 [cond-mat]*, February 2019. URL http://arxiv.org/abs/1902.08129. arXiv: 1902.08129.

Netsor program 3 Batchnorm (with Fully Connected Layers) over Multiple Batches

```
  // For $\phi : \mathbb{R} \to \mathbb{R}, h \in \mathbb{R}^B$,
  // let $\tilde{\phi}(h) = \phi((h - \nu(h))/\sigma(h))$,
  // where $\nu(h) = \frac{1}{B}\sum_{i=1}^B h_i$,
  // $\sigma(h) = \sqrt{\frac{1}{B}\sum_{i=1}^B (h_i - \nu(h))^2}$.
  // Let $\tilde{\phi}_i, i \in [B]$, be the ith coordinate of $\tilde{\phi}$.
  // Embeddings of $k$ batches of inputs
  // with ath batch having size $B_a$
Input: $\{W^1 x^{ia} : \mathsf{G}(n)\}_{a\in[k], i\in[B_a]}$
Input: $W^2, \dots, W^L : \mathsf{A}(n, n)$
  // Readout layer
Input: $v : \mathsf{G}(n)$
for $a \in [k]$ and $i \in [B_a]$ do
    $x^{1ia} := \tilde{\phi}_i(W^1 x^{1a}, \dots, W^1 x^{B_a a}) : \mathsf{H}(n)$
end for
for $l = 2, \dots, L$ do
    for $a \in [k]$ and $i \in [B_a]$ do
        $h^{lia} := W^l x^{l-1,ia} : \mathsf{G}(n)$
    end for
    for $a \in [k]$ and $i \in [B_a]$ do
        $x^{lia} := \tilde{\phi}_i(h^{l1a}, \dots, h^{lB_a a}) : \mathsf{H}(n)$
    end for
end for
Output: $\{v^\top x^{Lia}/\sqrt{n}\}_{a\in[k], i\in[B_a]}$
```

A Writing Standard Architectures in Netsor

In this section, we showcase example programs for batchnorm, skip connection, convolution, pooling, GRU/LSTM, and (scaled) attention. In most cases, we demonstrate the computation on a single input batch, image, or sequence. Generalization to multiple inputs is obvious and follows the pattern of Program 2. It should also be apparent that any composition of these gadgets can be expressed in Netsor .

Additionally, observe that all nonlinearities used in these programs are controlled (Definition 5.3) or parameter-controlled (Definition C.1), so that Theorem 5.4 or Theorem C.4 applies. Also we remark that the GP convergence results for batchnorm, GRU/LSTM (and RNNs in general), and attention are new.

Batchnorm, Fully-Connected Let $\tilde{\phi} : \mathbb{R}^B \to \mathbb{R}^B$,

$$\tilde{\phi}(h) \overset{\text{def}}{=} \phi\left(\frac{h - \nu(h)}{\sigma(h)}\right), \quad \text{where} \quad \nu(h) \overset{\text{def}}{=} \frac{1}{B}\sum_{i=1}^{B} h_i, \quad \sigma(h) \overset{\text{def}}{=} \sqrt{\frac{1}{B}\sum_{i=1}^{B}(h_i - \nu(h))^2}, \tag{4}$$

be batchnorm followed by coordinatewise nonlinearity ϕ, where $h \in \mathbb{R}^B$ should be interpreted as a single neuron across a batch, and ν and σ are the *batch* mean and standard deviations. Here, B should be thought of as fixed while $n \to \infty$. Then, given a batch of G-vars $y^1, \dots, y^B : \mathsf{G}(n)$ (for example, they could be the preactivations after applying a linear layer), we can express the application of batchnorm via `Nonlin` as

$$x^1 := \tilde{\phi}_1(y^1, \dots, y^B), \quad \dots, \quad x^B := \tilde{\phi}_B(y^1, \dots, y^B), \tag{5}$$

producing B H-vars $x^1, \dots, x^B : \mathsf{H}(n)$. Program 3 expresses more generally the computation of a batchnorm network on multiple batches and over multiple layers. Here, each "for-loop" is just shorthand for the corresponding unrolled program (since Definition 4.1 doesn't allow for-loops).

Skip Connection If $x : \mathsf{H}(n)$ is previous layer activation, and we have weights $W : \mathsf{A}(n, n)$ and bias $b : \mathsf{G}(n)$, then we can express skip connection as

$$\begin{aligned} \tilde{h} &:= Wx && \texttt{MatMul} \\ h &:= \tilde{h} + b && \texttt{LinComb} \\ \bar{x} &:= x + \phi(h) && \texttt{Nonlin} \end{aligned}$$

(Graph) Convolution Consider a convolution network with L layers, with layer l having n^l channels, with kernel positions ker^l (for example, for a 3×5 kernel, $ker^l = [3] \times [5]$), and with feature map pixel positions pos^l (for example, for a 128×256 feature map, $pos^l = [128] \times [256]$).

NETSOR program 4 2-Layer Convolutional Network with Global Average Pooling

```
Input: $\{W^1_j x_{i+j} : \mathsf{G}(n^1)\}_{j \in ker^1, i \in pos^1 \text{ s.t. } i+j \in pos^0}$
Input: $\{W^2_j : \mathsf{A}(n^2, n^1)\}_{j \in ker^2}$
  // Readout weights
Input: $v : \mathsf{G}(n^2)$
  // Layer 1 Convolution
  for $i \in pos^1$ do
      // Directly use input embeddings
      // LinComb
      // Sum is over all $j \in ker^1$ such that
      // there is $i' \in pos^0$ with $i' = j + i$
      $h^1_i := \sum_j W^1_j x_{i+j} : \mathsf{G}(n^1)$
      $x^1_i := \phi(h^1_i) : \mathsf{H}(n^1)$
  end for
  // Layer 2 Convolution
  for $j \in ker^2, i \in pos^1$ s.t. $i + j \in pos^2$ do
      // Convolution Weights Multiplication
      // MatMul
      $h^2_{i;j} := W^1_j x^1_{i+j} : \mathsf{G}(n^2)$
  end for
  for $i \in pos^2$ do
      // Sum is over all $j \in ker^2$ such that
      // there is $i' \in pos^1$ with $i' = j + i$
      $h^2_i := \sum_j h^2_{i;j} : \mathsf{G}(n^2)$
  end for
  // Nonlinearity & Global Average Pooling
  $\bar{x}^2 := \frac{1}{|pos^2|} \sum_{i \in pos^2} \phi(h^2_i) : \mathsf{H}(n^2)$
Output: $v^\top \bar{x}^2 / \sqrt{n^2}$
```

Then we can describe the weights of a stride-1 convolution as $\{W^l_{i\alpha\beta}\}_{l \in [L], i \in ker^l, \alpha \in [n^l], \beta \in [n^{l-1}]}$, so that for each $i \in ker^l$, $W^l_i \in \mathbb{R}^{n^l \times n^{l-1}}$ is a dense matrix. Further, suppose x is an image input to the network with pixel coordinates pos^0 and n^0 channels, $\{x_{i\alpha}\}_{i \in pos^0, \alpha \in [n^0]}$, so that x_i is a vector of dimension n^0. Then the application of the convolution W^1 on x is given by

$$h^1_i := (W^1 * x)_i = \sum_j W^1_j x_{i+j} \in \mathbb{R}^{n^1}, \quad h^1_{i\alpha} = (W^1 * x)_{i\alpha} = \sum_{j\beta} W^1_{j\alpha\beta} x_{i+j,\beta} \in \mathbb{R},$$

where the sums are over $j \in ker^1, i + j \in pos^0$, and $\beta \in [n^0]$. For higher layers we have similar formulas, with the only difference that we treat $W^1_j x_{i+j}$ as input G-vars but treat $W^{l+1}_j x^l_{i+j}, l \geq 2$, as a `MatMul` operation. Thus, our framework can express CNN computations by expressing the individual matrix multiplications $W_j x_{i+j}$ and reusing the matrices W_j. Program 4 shows a simple example program, and Program 7 shows a full-fledged example over multiple inputs. Again, each "for-loop" is just shorthand for the corresponding unrolled program. Higher stride and general graph convolution can be expressed likewise.

Pooling We continue the notational scheme of the exposition on convolutions above. Given the feature maps of layer l, $\{x_{i\alpha}\}_{i \in pos^l, \alpha \in [n^l]}$, global average pooling produces a single vector $\bar{x} = \{\bar{x}_\alpha\}_{\alpha \in [n^l]}$ given by

$$\bar{x} := \frac{1}{|pos^l|} \sum_{i \in pos^l} x_i, \qquad \text{using } \texttt{LinComb}.$$

See Program 4 for an example in combination with convolutional layers. We can similarly express max pooling. Suppose $pos^l = [2k] \times [2k]$. Then max pooling with, for example, 2×2 kernel and stride of 2 would produce $\{\hat{x}_{j\alpha}\}_{j \in [k] \times [k], \alpha \in [n^l]}$, with

$$\hat{x}_{j\alpha} := \max(\{x_{2j+i,\alpha}\}_{i \in \{0,1\} \times \{0,1\}, 2j+i \in pos^l}), \qquad \text{using } \texttt{Nonlin}.$$

GRU and LSTM We present a program expressing GRU computation in Program 5; the program for LSTM is similar and is omitted. The overall pattern is similar to the program for simple RNN (Program 2), but with a crucial subtlety regarding the gating mechanism. In Program 5, $\tilde{z}^s, \tilde{r}^s, \tilde{h}^s$ are G-vars, but the gates $\sigma(\tilde{z}^s), \sigma(\tilde{r}^s)$ (where σ is sigmoid) and the candidate update $\phi(\tilde{h}^s)$ are not G-vars. As we can only apply `Nonlin` to G-vars, this requires us to unroll the definition of h^s to be a function of only G-vars. However, Appendix E presents expanded, but semantically equivalent versions of NETSOR which allow a more succinct representation of the same computation; see Program 11. Finally, Program 8 presents a full-fledged program with multiple input sequences.

NETSOR program 5 GRU, with Gating Function σ and Activation Function ϕ

```
  // Embeddings of input sequence
Input: $U_z x^1, \dots, U_z x^T : \mathsf{G}(n)$
Input: $U_r x^1, \dots, U_r x^T : \mathsf{G}(n)$
Input: $U_h x^1, \dots, U_h x^T : \mathsf{G}(n)$
  // Parameters
Input: $W_z, W_r, W_h : \mathsf{A}(n, n)$
Input: $b_z, b_r, b_h : \mathsf{G}(n)$
  // Initial GRU state
Input: $h^0 : \mathsf{G}(n)$
  // Readout layer
Input: $v : \mathsf{G}(n)$
  // Time step 1
  $h_z^1 := W_z h^0 : \mathsf{G}(n)$
  $\tilde{z}^1 := h_z^1 + U_z x^1 + b_z : \mathsf{G}(n)$
  $h_r^1 := W_r h^0 : \mathsf{G}(n)$
  $\tilde{r}^1 := h_r^1 + U_r x^1 + b_r : \mathsf{G}(n)$
  // $\sigma$ is gating function, typically sigmoid; applying Nonlin
  $\hat{h}^0 := h^0 \odot \sigma(\tilde{r}^1) : \mathsf{H}(n)$
  $h_h^1 := W_h \hat{h}^0 : \mathsf{G}(n)$
  $\tilde{h}^1 := h_h^1 + U_h x^1 + b_h : \mathsf{G}(n)$
  // Apply Nonlin
  // $\phi$ is activation function, typically $\tanh$
  $h^1 := (1 - \sigma(\tilde{z}^1)) \odot h^0 + \sigma(\tilde{z}^1) \odot \phi(\tilde{h}^1) : \mathsf{H}(n)$
  // Time step 2
  $h_z^2 := W_z h^1 : \mathsf{G}(n)$
  $\tilde{z}^2 := h_z^2 + U_z x^2 + b_z : \mathsf{G}(n)$
  $h_r^2 := W_r h^1 : \mathsf{G}(n)$
  $\tilde{r}^2 := h_r^2 + U_r x^2 + b_r : \mathsf{G}(n)$
  // Morally, $\hat{h}^1 = \sigma(\tilde{r}^1) \odot h^1$, but we need to unroll $h^1$ to apply Nonlin
  // This can be expressed with more brevity using NETSOR∘ ; see Remark E.5
  $\hat{h}^1 := \sigma(\tilde{r}^1) \odot ((1 - \sigma(\tilde{z}^1)) \odot h^0 + \sigma(\tilde{z}^1) \odot \phi(\tilde{h}^1)) : \mathsf{H}(n)$
  $h_h^2 := W_h \hat{h}^1 : \mathsf{G}(n)$
  $\tilde{h}^2 := h_h^2 + U_h x^2 + b_h : \mathsf{G}(n)$
  // Unrolling $h^2$ to a coordinatewise function of G-vars
  $h^2 := (1 - \sigma(\tilde{z}^2)) \odot (1 - \sigma(\tilde{z}^1)) \odot h^0 + (1 - \sigma(\tilde{z}^2)) \odot \sigma(\tilde{z}^1) \odot \phi(\tilde{h}^1) + \sigma(\tilde{z}^2) \odot \phi(\tilde{h}^2) : \mathsf{H}(n)$
  // Time step 3
  $\vdots$
  // Time step $T$
  // Define $\tilde{z}^T, \tilde{r}^T, \tilde{h}^T$ just like above
  $\vdots$
  // Unrolling $h^T$ to a coordinatewise function of G-vars
  $h^T := h^0 \odot \bigodot_{i=1}^T (1 - \sigma(\tilde{z}^i)) + \sum_{j=1}^T \phi(\tilde{h}^j) \odot \sigma(\tilde{z}^j) \odot \bigodot_{l=j+1}^T (1 - \sigma(\tilde{z}^l)) : \mathsf{H}(n)$
Output: $(v^\top h^1/\sqrt{n}, \dots, v^\top h^T/\sqrt{n})$
```

Layernorm Layernorm requires the extended rule Nonlin$^+$ to express. Recall for $x \in \mathbb{R}^n$ (thought of as the vector of activations with one entry per neuron in a layer; contrast with batchnorm), $\text{Layernorm}(x) = \frac{x - \nu(x)}{\sigma(x)}$ where $\nu(x) = \frac{1}{n}\sum_\alpha x_\alpha$ and $\sigma(x) = \sqrt{\frac{1}{n}\sum_{\alpha=1}^n (x_\alpha - \nu(x))^2}$. As we will see, $\nu(x)$ and $\sigma(x)$ will both converge a.s. to a deterministic limit. Thus $\text{Layernorm}(x)$ is just a linear combination of x with the constant-1s vector (considered as a input G-var), with (roughly deterministic) coefficients $\mu(x)$ and $\sigma(x)$. This is expressible using the Nonlin$^+$ rule:

$$\text{Layernorm}(x) := \psi(x; \nu(x), \sigma(x)), \quad \text{where} \quad \psi(x; \theta_1, \theta_2) \overset{\text{def}}{=} \frac{x - \theta_1}{\theta_2}.$$

Similarly, if layernorm is followed up by a nonlinearity ϕ, then we can express

$$\phi(\text{Layernorm}(x)) := \psi(x; \nu(x), \sigma(x)), \quad \text{where} \quad \psi(x; \theta_1, \theta_2) \overset{\text{def}}{=} \phi\left(\frac{x - \theta_1}{\theta_2}\right).$$

If layernorm is preceded by a nonlinearity, then likewise we can write

$$\text{Layernorm}(\phi(x)) := \psi(x; \nu(\phi(x)), \sigma(\phi(x))), \quad \text{where} \quad \psi(x; \theta_1, \theta_2) \overset{\text{def}}{=} \frac{\phi(x) - \theta_1}{\theta_2}.$$

Scaled Attention Scaled attention requires the extended rule Nonlin$^+$ to express. Consider the following version of scaled attention: Given a query vector $q \in \mathbb{R}^n$, keys $k^i \in \mathbb{R}^n$ for each $i \in [r]$, and corresponding values $v^i \in \mathbb{R}^m$, $i \in [r]$, we can define the following scaled attention

$$\text{Attention}(q, \{k^i\}_i, \{v^i\}_i) \overset{\text{def}}{=} \sum_{i=1}^{r} a_i v^i, \qquad a_i \overset{\text{def}}{=} \text{SoftMax}(q^\top k^1/n, \ldots, q^\top k^r/n)_i.$$

If q, k^i are given as H-vars in a NETSOR program, then Theorem 5.4 will show that $q^\top k^i/n$ converges almost surely to a deterministic limit, so that each a_i converges likewise. If each $v^i = \psi(g^i)$ for some G-var g^i and fixed nonlinearity ψ, then attention can be expressed as follows using Nonlin$^+$:

$$\text{Attention}(q, \{k^i\}_i, \{v^i\}_i) = \phi(g^1, \ldots, g^r; a_1, \ldots, a_r)$$

where

$$\phi(x^1, \ldots, x^r; \theta_1, \ldots, \theta_r) \overset{\text{def}}{=} \theta_1 \psi(x^1) + \cdots + \theta_r \psi(x^r).$$

More complicated variants, such as allowing ψ to take multiple G-vars as inputs, is likewise expressible. When used as part of a decoder, a mask needs to be placed on the pre-softmax values so that no attention is paid to tokens *in the future*. This is aptly called *masked attention* and is given by the following formula: for $j \in [r]$,

$$\text{MaskedAttention}_j(q, \{k^i\}_{i=1}^r, \{v^i\}_{i=1}^r) = \sum_{i=1}^{r} a_i^j v^i,$$
$$\text{where} \quad a_i^j = \text{SoftMax}(q^\top k^1/n, \ldots, q^\top k^j/n, -\infty, \ldots, -\infty)_i.$$

It can obviously be expressed in NETSOR in the same fashion as (non-masked) attention.

Note that Vaswani et al. [55] scales the pre-softmax values by $1/\sqrt{n}$ instead of $1/n$:

$$a_i = \text{SoftMax}(q^\top k^1/\sqrt{n}, \ldots, q^\top k^r/\sqrt{n})_i.$$

This is useful if $q_\alpha k_\alpha^i$ has mean zero (averaged over α) for each i, so that $q^\top k^i/\sqrt{n}$ becomes roughly Gaussian, whereas $q^\top k^i/n$ converges to 0. However, when the zero-mean condition doesn't hold, $q^\top k^i/\sqrt{n}$ would only blow up to infinity.

B Example GP Kernel Computation with NETSOR

In this section, we show how to compute the GP kernel of different architecture, following the recursive construction of Theorem 5.4.

First, we review the *V-transform* of a nonlinearity.

Definition B.1. Given a multivariate nonlinearity $\Phi : \mathbb{R}^B \to \mathbb{R}^B$, its *V-transform* V_Φ is a function taking $B \times B$ positive semidefinite matrices to $B \times B$ positive semidefinite matrices, and is given by the following formula

$$\mathrm{V}_\Phi(K) \overset{\text{def}}{=} \mathop{\mathbb{E}}_{z \sim \mathcal{N}(0,K)} \Phi(z)\Phi(z)^\top.$$

When $\phi : \mathbb{R} \to \mathbb{R}$, we take V_ϕ to be V-transform of the $\mathbb{R}^B \to \mathbb{R}^B$ function that applies ϕ to each coordinate.

We collect below some of the common V-transforms. Here we describe the V-transforms using the function notation of kernels, but we shall freely switch between the function notation and the matrix notation in what follows.

Fact B.2 ([11]). *For any kernel K,*

$$\mathrm{V}_{\mathrm{ReLU}}(K)(x,x') = \frac{1}{2\pi}(\sqrt{1-c^2} + (\pi - \arccos c)c)\sqrt{K(x,x)K(x',x')}$$
$$\mathrm{V}_{\mathrm{ReLU}'}(K)(x,x') = \frac{1}{2\pi}(\pi - \arccos c)$$

where $c = K(x,x')/\sqrt{K(x,x)K(x',x')}$.

Fact B.3 ([41]). *For any kernel K,*

$$\mathrm{V}_{\mathrm{erf}}(K)(x,x') = \frac{2}{\pi}\arcsin\frac{K(x,x')}{\sqrt{(K(x,x)+0.5)(K(x',x')+0.5)}}$$
$$\mathrm{V}_{\mathrm{erf}'}(K)(x,x') = \frac{4}{\pi\sqrt{(1+2K(x,x))(1+2K(x',x'))-4K(x,x')^2}}.$$

Fact B.4. *Let $\phi(x) = \exp(x/\sigma)$ for some $\sigma > 0$. For any kernel K,*

$$\mathrm{V}_{\phi}(K)(x,x') = \exp\left(\frac{K(x,x)+2K(x,x')+K(x',x')}{2\sigma^2}\right).$$

In Appendix B.3, we will also discuss the V-transform of batchnorm (which has been derived in [65]).

B.1 MLP

The kernel computation of a multilayer perceptron is by now well-known [46, 50, 37]. In this section, we work out an example of how to recover the usual kernel computation via tensor programs.

B.1.1 MLP with Single Input (Program 1)

The aim here is to illustrate step-by-step applications of Eq. (2) for Program 1, but note that in practice, it is often more convenient to find some bulk recursive or compositional structure of Σ, and to leverage that structure for computing Σ (see Appendix B.2 for an example).

For simplicity, assume the nonlinearities ϕ are ReLUs and the widths $n^1 = n^2 = n$ to satisfy Assumption 5.1. By Corollary 5.5, we know that the output of the MLP, $v^T x^2/\sqrt{n}$, is distributed as a Gaussian with mean 0 and variance of $\sigma^2 \mathbb{E}_z \phi^2(z)$, where $z \sim \mathcal{N}(\mu(h^2), \Sigma(h^2,h^2))$, and h^2 is the layer 2 preactivation (also a G-var) in Program 1. Therefore, all we need is to compute $\mu(h^2)$ and $\Sigma(h^2,h^2)$ using (2), which requires the calculation of μ and Σ for possibly all the other G-vars in Program 1 due to the recursive nature of (2). In this example, we shall compute all entries of μ and Σ explicitly as a demonstration. We do so for each G-var in the order of appearance in the NETSOR program. Explicitly, the ordering is $\mathcal{G} = \left(W^1x, b^1, b^2, v, h^1, \tilde{h}^2, h^2\right)$. The input G-vars are W^1x, b^1, b^2, and v, and the sole input A-var is W^2.

Setup In the fashion of typical Glorot initializations, we shall sample the parameters W^1, W^2, b^1, b^2 of the network as

$$W^1_{\alpha\beta} \sim \mathcal{N}(0, 1/\dim x), W^2_{\alpha\beta} \sim \mathcal{N}(0,1/n), v_\alpha \sim \mathcal{N}(0,1), b^1_\alpha \sim \mathcal{N}(0,1), b^2_\alpha \sim \mathcal{N}(0,1).$$

This corresponds, in the context of Assumption 5.1, to setting $\sigma_{W^2} = 1$, $\mu^{\text{in}} = 0$ (in particular, $\mu(W^1x) = \mu^{\text{in}}(W^1x) = 0$ due to the sampling of W^1), and Σ^{in} as follows:

$$
\begin{aligned}
&\Sigma^{\text{in}}(W^1x, W^1x) = 1\\
&\Sigma^{\text{in}}(b^1, b^1) = 1\\
&\Sigma^{\text{in}}(b^2, b^2) = 1\\
&\Sigma^{\text{in}}(b^2, b^1) = 0\\
&\Sigma^{\text{in}}(v, v) = 1\\
&\Sigma^{\text{in}}(b^i, W^1x) = 0 \text{ for } i \in \{1,2\}\\
&\Sigma^{\text{in}}(b^i, v) = 0 \text{ for } i \in \{1,2\}\\
&\Sigma^{\text{in}}(v, W^1x) = 0
\end{aligned}
$$

Calculating μ and Σ Now we will show a detailed calculation of μ and Σ for all of the G-vars $\mathcal{G}$ appearing here. By the input G-var case of Eq. (2),

$$
\begin{aligned}
&\Sigma(W^1x, W^1x) = \Sigma^{\text{in}}(W^1x, W^1x) = 1\\
&\Sigma(b^1, b^1) = \Sigma^{\text{in}}(b^1, b^1) = 1\\
&\Sigma(b^2, b^2) = \Sigma^{\text{in}}(b^2, b^2) = 1\\
&\Sigma(b^1, b^2) = \Sigma^{\text{in}}(b^2, b^1) = 0\\
&\Sigma(v, v) = \Sigma^{\text{in}}(v, v) = 1\\
&\Sigma(W^1x, b^i) = \Sigma(b^i, W^1x) = \Sigma^{\text{in}}(b^i, W^1x) = 0 \text{ for } i \in \{1,2\}\\
&\Sigma(v, b^i) = \Sigma(b^i, v) = \Sigma^{\text{in}}(b^i, v) = 0 \text{ for } i \in \{1,2\}\\
&\Sigma(W^1x, v) = \Sigma(v, W^1x) = \Sigma^{\text{in}}(v, W^1x) = 0
\end{aligned}
$$

Next, we extend μ and Σ to h^1, introduced via `LinComb` by $h^1 := W^1x + b^1$. Note that h^1 is a G-var of the form $g = \sum_i a_i y^i$ where $a_1 = a_2 = 1$, $y^1 = W^1x$, and $y^2 = b^1$. Therefore, by the `LinComb` case of Eq. (2),

$$
\begin{aligned}
&\mu(h^1) = \mu(W^1x) + \mu(b^1) = 0\\
&\Sigma(h^1, W^1x) = \Sigma(W^1x, W^1x) + \Sigma(b^1, W^1x) = 1 + 0 = 1\\
&\Sigma(h^1, b^1) = \Sigma(W^1x, b^1) + \Sigma(b^1, b^1) = 0 + 1 = 1\\
&\Sigma(h^1, b^2) = \Sigma(W^1x, b^2) + \Sigma(b^1, b^2) = 0 + 0 = 0\\
&\Sigma(h^1, v) = \Sigma(W^1x, v) + \Sigma(b^1, v) = 0 + 0 = 0\\
&\Sigma(h^1, h^1) = \Sigma(h^1, W^1x) + \Sigma(h^1, b^1) = 1 + 1 = 2.
\end{aligned}
$$

So h^1 is correlated with W^1x and b^1 in obvious ways, and is independent from b^2 and v.

Next, we extend μ and Σ to, introduced via `MatMul` by $\tilde{h}^2 := W^2x^1$. Note that $\tilde{h}^2$ is a G-var of the form $g = Wh$ where $W = W^2$ is an A-var, and $h = x^1$ is an H-var introduced by $x^1 := \text{ReLU}(h^1)$. Therefore, by the "otherwise" case of Eq. (2),

$$
\begin{aligned}
&\mu(\tilde{h}^2) = 0\\
&\Sigma(\tilde{h}^2, W^1x) = 0\\
&\Sigma(\tilde{h}^2, b^1) = 0\\
&\Sigma(\tilde{h}^2, b^2) = 0\\
&\Sigma(\tilde{h}^2, v) = 0\\
&\Sigma(\tilde{h}^2, h^1) = 0
\end{aligned}
$$

and by the `MatMul` case of Eq. (2) (setting ϕ and $\bar{\phi}$ there to both be ReLU),

$$
\Sigma(\tilde{h}^2, \tilde{h}^2) = \sigma_{W^2}^2 \mathop{\mathbb{E}}_z \phi(z)\bar{\phi}(z) = \mathop{\mathbb{E}}_z \text{ReLU}(z)^2 = 1
$$
$$
\text{where } z \sim \mathcal{N}(\mu(h^1), \Sigma(h^1, h^1)) = \mathcal{N}(0, 2).
$$

Netsor program 6 MLP Computation on a Set of Inputs

```
    // Embeddings of inputs
Input: $W^1x^1, \dots, W^1x^B : \mathsf{G}(n)$
    // Biases across $L$ layers
Input: $b^1, \dots, b^L : \mathsf{G}(n)$
    // Weights from layer 2 on
Input: $W^2, \dots, W^L : \mathsf{A}(n, n)$
    // Readout weights
Input: $v : \mathsf{G}(n)$
for $i = 1, \dots, B$ do
    $h^{1i} := W^1x^i + b^1 : \mathsf{G}(n)$
    $x^{1i} := \phi(h^{1i}) : \mathsf{H}(n)$
    for $l = 2, \dots, L$ do
        $\tilde{h}^{li} := W^l x^{l-1,i} : \mathsf{G}(n)$
        $h^{li} := \tilde{h}^{li} + b^l : \mathsf{G}(n)$
        $x^{li} := \phi(h^{li}) : \mathsf{H}(n)$
    end for
end for
Output: $(v^\top x^{L1}/\sqrt{n}, \dots, v^\top x^{LB}/\sqrt{n})$
```

Thus, $\tilde{h}^2$ can be thought of as "independent" from all other G-vars.

Finally, we extend μ and Σ to h^2, introduced via `LinComb` by $h^2 := \tilde{h}^2 + b^2$. Note that h^2 is a G-var of the form $g = \sum_i a_i y^i$ where $a_1 = a_2 = 1$, $y^1 = \tilde{h}^2$, and $y^2 = b^2$. Then by the `LinComb` case of Eq. (2),

$$\begin{aligned}
&\mu(h^2) = \mu(\tilde{h}^2) + \mu(b^2) = 0\\
&\Sigma(h^2, W^1x) = \Sigma(\tilde{h}^2, W^1x) + \Sigma(b^2, W^1x) = 0 + 0 = 0\\
&\Sigma(h^2, b^1) = \Sigma(\tilde{h}^2, b^1) + \Sigma(b^2, b^1) = 0 + 0 = 0\\
&\Sigma(h^2, b^2) = \Sigma(\tilde{h}^2, b^2) + \Sigma(b^2, b^2) = 0 + 1 = 1\\
&\Sigma(h^2, v) = \Sigma(\tilde{h}^2, v) + \Sigma(b^2, v) = 0 + 0 = 0\\
&\Sigma(h^2, h^1) = \Sigma(\tilde{h}^2, h^1) + \Sigma(b^2, h^1) = 0\\
&\Sigma(h^2, \tilde{h}^2) = \Sigma(\tilde{h}^2, \tilde{h}^2) + \Sigma(b^2, \tilde{h}^2) = 1 + 0 = 1\\
&\Sigma(h^2, h^2) = \Sigma(\tilde{h}^2, h^2) + \Sigma(b^2, h^2) = \Sigma(h^2, \tilde{h}^2) + \Sigma(h^2, b^2) = 1 + 1 = 2.
\end{aligned}$$

Note that h^2 turns out to be "independent" from h^1, i.e. $\Sigma(h^2, h^1) = 0$, just as one might expect from the mean field or the NNGP literature.

Distribution of the Program Output We are now done with calculating $\mu(g)$ and $\Sigma(g, g')$ for all $g, g' \in \mathcal{G}$. Recall the output of the program is $v^\top x^2/\sqrt{n}$, where x^2 was introduced via `Nonlin` by $x^2 := \mathrm{ReLU}(h^2)$. According to Corollary 5.5, the output variance is then given by

$$\begin{aligned}
&\sigma_v^2 \mathop{\mathbb{E}}_z \phi^2(z) = \mathop{\mathbb{E}}_z \mathrm{ReLU}(z)^2 = 1,\\
&\text{where } z \sim \mathcal{N}(\mu(h^2), \Sigma(h^2, h^2)) = \mathcal{N}(0, 2).
\end{aligned}$$

B.1.2 MLP with Multiple Inputs (Program 6)

Now suppose we have an L-layer MLP with B inputs $x^1, \dots, x^B$. Program 6 expresses its computation (again, the "for" loops are shorthands for the unrolled series of assignments).

Here we will avoid computing out all values of Σ but only those that affect the infinite-width GP. By Corollary 5.5, the output of Program 6 is distributed as

$$K_{ij} \overset{\text{def}}{=} \mathrm{Cov}\left(\frac{v^\top x^{Li}}{\sqrt{n}}, \frac{v^\top x^{Lj}}{\sqrt{n}}\right) = \sigma_v^2 \mathop{\mathbb{E}}_Z \phi(Z^{h^{Li}})\phi(Z^{h^{Lj}}) \tag{6}$$

where σ_v^2 is the coordinatewise variance of v, $Z \sim \mathcal{N}(\mu, \Sigma)$, and $Z^{h^{Li}}$ is the component of Z corresponding to h^{Li} (likewise for $Z^{h^{Lj}}$). Therefore, we need to compute μ and Σ for the G-vars $\{h^{L1}, \dots, h^{LB}\}$.

Setup Suppose the inputs x^i have dimension m. If we sample the neural network parameters in the usual Glorot fashion,

$$\begin{aligned} W^1_{\alpha\beta} &\sim \mathcal{N}(0, \sigma_w^2/m), \quad & W^l_{\alpha\beta} &\sim \mathcal{N}(0, \sigma_w^2/n),\ \forall 2 \le l \le L, \\ v_\alpha &\sim \mathcal{N}(0, \sigma_v^2), & b^l_\alpha &\sim \mathcal{N}(0, \sigma_b^2),\ \forall l \in [L], \end{aligned} \tag{7}$$

then we have Σ^{in} defined by

$$\Sigma^{\text{in}}(W^1 x^i, W^1 x^j) = \sigma_w^2 x^{i\top} x^j / m, \quad \Sigma^{\text{in}}(b^l, b^l) = \sigma_b^2, \quad \Sigma^{\text{in}}(v, v) = \sigma_v^2$$

for all $i, j \in [B]$ and $l \in [L]$, and $\Sigma^{\text{in}}(g, g') = 0$ for any other pairs of input G-vars g, g'. On the other hand, $\mu^{\text{in}}(g) = 0$ for all input G-vars g.

Computing μ From Eq. (2), it is clear that $\mu^{\text{in}} = 0$ implies $\mu = 0$.

Computing Σ Again, our goal is to compute Σ restricted to the G-vars $\{h^{L1}, \ldots, h^{LB}\}$.

Lemma B.5. *For any $l = 2, \ldots, L$ and any $i, j \in [B]$,*

$$\Sigma(h^{li}, h^{lj}) = \sigma_w^2 \mathop{\mathbb{E}}_{z_1, z_2} \phi(z_1)\phi(z_2) + \sigma_b^2 \tag{8}$$

where $(z_1, z_2) \sim \mathcal{N}\left(0, \Sigma|_{h^{l-1,i}, h^{l-1,j}}\right)$, and (unrolling the restriction notation)

$$\Sigma|_{h^{l-1,i}, h^{l-1,j}} = \begin{pmatrix} \Sigma(h^{l-1,i}, h^{l-1,i}) & \Sigma(h^{l-1,i}, h^{l-1,j}) \\ \Sigma(h^{l-1,j}, h^{l-1,i}) & \Sigma(h^{l-1,j}, h^{l-1,j}) \end{pmatrix}.$$

Proof. From Program 6, h^{li} is introduced via `LinComb` by

$$h^{li} := \tilde{h}^{li} + b^l.$$

Thus by the `LinComb` cases of Eq. (2), we have

$$\begin{aligned} \Sigma(h^{li}, h^{lj}) &= \Sigma(\tilde{h}^{li}, h^{lj}) + \Sigma(b^l, h^{lj}) \\ &= \Sigma(\tilde{h}^{li}, \tilde{h}^{lj}) + \Sigma(b^l, \tilde{h}^{lj}) + \Sigma(\tilde{h}^{li}, b^l) + \Sigma(b^l, b^l). \end{aligned}$$

Now, we cannot pattern match $\Sigma(b^l, \tilde{h}^{lj})$ with any of the cases of Eq. (2) other than the "otherwise" case, which means $\Sigma(b^l, \tilde{h}^{lj}) = 0$. Likewise, $\Sigma(\tilde{h}^{li}, b^l) = 0$. Therefore,

$$\begin{aligned} \Sigma(h^{li}, h^{lj}) &= \Sigma(\tilde{h}^{li}, \tilde{h}^{lj}) + \Sigma(b^l, b^l) \\ &= \Sigma(\tilde{h}^{li}, \tilde{h}^{lj}) + \sigma_b^2. \end{aligned}$$

Now let's analyze the $\Sigma(\tilde{h}^{li}, \tilde{h}^{lj})$ term. The G-var $\tilde{h}^{li}$ is introduced via `MatMul` by

$$\tilde{h}^{li} := W^l x^{l-1,i}, \quad \text{where} \quad x^{l-1,i} := \phi(h^{l-1,i})$$

and likewise for h^{lj}. By the `MatMul` case of Eq. (2), we have

$$\Sigma(\tilde{h}^{li}, \tilde{h}^{lj}) = \sigma_w^2 \mathop{\mathbb{E}}_{Z} \phi(Z^{h^{l-1,i}}) \phi(Z^{h^{l-1,j}})$$

where $Z \sim \mathcal{N}(\mu, \Sigma)$. Since the integrand only depends two components of Z, we can simplify the expression as

$$\mathop{\mathbb{E}}_{Z} \phi(Z^{h^{l-1,i}}) \phi(Z^{h^{l-1,j}}) = \mathop{\mathbb{E}}_{z_1, z_2} \phi(z_1)\phi(z_2), \quad \text{where} \quad (z_1, z_2) \sim \mathcal{N}\left(0, \Sigma|_{h^{l-1,i}, h^{l-1,j}}\right).$$

Putting it all together, we recover the expression in the claim, as desired. □

Computing the GP Kernel K Eq. (8) along with Eq. (6) gives all we need to compute K by recursion. If ϕ has a nice V-transform V_ϕ, then we can vectorize this equation and obtain the following algorithm (which, again, is by now well-known [46, 50, 37])

Computing MLP kernel on B inputs

Consider an L-layer MLP with nonlinearity ϕ at each layer. Suppose we have B network inputs $x^1, \ldots, x^B \in \mathbb{R}^m$, as in Program 6, and we sample the MLP parameters as in Eq. (7). Then the MLP converges in distribution to a GP on those B inputs, with kernel K computed as follows

1. Initialize $K \in \mathbb{R}^{B\times B}$ by $K_{ij} \leftarrow \sigma_w^2 x^{i\top} x^j / m$
2. For $l = 1, \ldots, L-1$, do $K \leftarrow \sigma_w^2 \mathrm{V}_\phi(K) + \sigma_b^2$
3. Return $K \leftarrow \sigma_v^2 \mathrm{V}_\phi(K)$

B.2 Simple RNN (Program 2)

By Corollary 5.5, we know that the output of Program 2,

$$\left(\frac{v^\top s^{11}}{\sqrt{n}}, \ldots, \frac{v^\top s^{t1}}{\sqrt{n}}, \frac{v^\top s^{12}}{\sqrt{n}}, \ldots, \frac{v^\top s^{r2}}{\sqrt{n}}\right)$$

is, in the large n limit, distributed as a Gaussian with mean 0 and the covariance K where, for any $a, b \in \{1, 2\}$ (denoting sequence number),

$$K_{ia,jb} \overset{\text{def}}{=} \lim_{n\to\infty} \mathrm{Cov}\left(\frac{v^\top s^{ia}}{\sqrt{n}}, \frac{v^\top s^{jb}}{\sqrt{n}}\right) = \sigma_v^2 \mathop{\mathbb{E}}_Z \phi(Z^{h^{ia}})\phi(Z^{h^{jb}})$$

where $Z \sim \mathcal{N}(\mu, \Sigma)$, and $Z^{h^{ia}}$ is the component of Z corresponding to h^{ia} and likewise for $Z^{h^{jb}}$. Therefore, we need to compute μ and Σ for the G-vars $\{h^{11}, \ldots, h^{s1}, h^{12}, \ldots, h^{t2}\}$ in Program 2.

Setup Suppose the input tokens x^{ia} to the RNN have dimension m. We will obtain the μ and Σ for Program 2 with

$$U_{\alpha\beta} \sim \mathcal{N}(0, \sigma_U^2/m), W_{\alpha\beta} \sim \mathcal{N}(0, \sigma_W^2/n), b_\alpha \sim \mathcal{N}(0, \sigma_b^2), v_\alpha \sim \mathcal{N}(0, \sigma_v^2). \tag{9}$$

The randomization of U induces the following covariance structure in the input token embeddings

$$\Sigma^{\text{in}}(Ux, Uy) = \sigma_U^2 x^\top y / m \tag{10}$$

for any $x, y \in \{x^{i1}\}_{i=1}^t \cup \{x^{j2}\}_{j=1}^r$. For any other pair g, g' of input G-vars, the sampling implies $\Sigma^{\text{in}}(g, g') = 0$. Additionally, $\mu^{\text{in}}(g) = 0$ for all input G-var g.

Computing μ In fact, one can quickly see that $\mu(g) = 0$ for all G-vars g, not just the input G-vars.

Computing Σ By Eq. (2), all $\{h^{i1}, \tilde{h}^{i1}\}_{i=1}^t \cup \{h^{j2}, \tilde{h}^{j2}\}_{j=1}^r$ are possibly correlated with each other. They satisfy the following recurrence

Lemma B.6. *For any $a, b \in \{1, 2\}$, we have*

$$\Sigma(h^{ia}, h^{jb}) = \Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) + \Sigma^{\text{in}}(Ux^{ia}, Ux^{jb}) + \sigma_b^2, \qquad \forall i, j \geq 1 \tag{11}$$

$$\Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) = \sigma_W^2 \mathbb{E}\, \phi(z_1)\phi(z_2), \qquad \forall i, j \geq 2, \tag{12}$$

where

$$(z_1, z_2) \sim \mathcal{N}(0, \Sigma|_{h^{i-1,a}, h^{j-1,b}}),$$

and the base case $\Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) = 0$ if $i \leq 1$ or $j \leq 1$. Here, $\Sigma|_{set}$ means the submatrix of Σ with rows and columns in set.

Note that the notation b appears both as a sequence index and as the bias of the RNN. Since the former will only appear as a superscript and the latter will not, there should be no confusion.

Proof. Note that for each $i \geq 2$, h^{ia} is introduced via `LinComb` by $h^{ia} := \tilde{h}^{ia} + Ux^{ia} + b$, where $\tilde{h}^{ia}$, Ux^{ia}, and b are G-vars. Then by the `LinComb` case of Eq. (2),

$$\begin{aligned}\Sigma(h^{ia}, h^{jb}) &= \Sigma(\tilde{h}^{ia}, h^{jb}) + \Sigma(Ux^{ia}, h^{jb}) + \Sigma(b, h^{jb}) \\ &= \Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) + \Sigma(Ux^{ia}, \tilde{h}^{jb}) + \Sigma(b, \tilde{h}^{jb}) \\ &\quad + \Sigma(\tilde{h}^{ia}, Ux^{jb}) + \Sigma(Ux^{ia}, Ux^{jb}) + \Sigma(b, Ux^{jb}) \\ &\quad + \Sigma(\tilde{h}^{ia}, b) + \Sigma(Ux^{ia}, b) + \Sigma(b, b).\end{aligned}$$

By the “otherwise” case of Eq. (2),

$$\Sigma(Ux^{ia}, \tilde{h}^{jb}) = \Sigma(b, \tilde{h}^{jb}) = \Sigma(\tilde{h}^{ia}, Ux^{jb}) = \Sigma(\tilde{h}^{ia}, b) = 0,$$

and by the “input G-var” case of Eq. (2),

$$\Sigma(b, Ux^{jb}) = \Sigma(\tilde{h}^{ia}, b) = \Sigma^{\text{in}}(b, Ux^{jb}) = \Sigma^{\text{in}}(\tilde{h}^{ia}, b) = 0.$$

We thus have

$$\begin{aligned}\Sigma(h^{ia}, h^{jb}) &= \Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) + \Sigma(Ux^{ia}, Ux^{jb}) + \Sigma(b, b) \\ &= \Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) + \Sigma(Ux^{ia}, Ux^{jb}) + \sigma_b^2\end{aligned}$$

which is Eq. (11).

Next, note that $\tilde{h}^{ia}$ is introduced via `MatMul` by $\tilde{h}^{ia} := Ws^{ia}$, and s^{ia} is an H-var introduced by $s^{ia} := \phi(h^{ia})$. Thus, by the `MatMul` rule of Eq. (2),

$$\Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) = \sigma_W^2 \mathop{\mathbb{E}}_{Z} \phi(Z^{h^{ia}})\phi(Z^{h^{jb}})$$

where $Z \sim \mathcal{N}(\mu, \Sigma)$ and $Z^{h^{ia}}$ (resp. $Z^{h^{jb}}$) is its component corresponding to h^{ia} (resp. h^{jb}). Since the integrand only depends on the two components $Z^{h^{ia}}$ and $Z^{h^{jb}}$, we can rewrite

$$\Sigma(\tilde{h}^{ia}, \tilde{h}^{jb}) = \sigma_W^2 \mathop{\mathbb{E}}_{z_1, z_2} \phi(z_1)\phi(z_2)$$

where $(z_1, z_2) \sim \mathcal{N}(0, \Sigma|_{h^{i-1,a}, h^{j-1,b}})$. This is Eq. (12). □

Let us formulate the results above in a way more suggestive of the algorithm required to compute the kernel. For any $2 \leq p \leq t, 2 \leq q \leq r$, define $\tilde{H}^{pq} \stackrel{\text{def}}{=} \{\tilde{h}^{i1}\}_{i=2}^{p} \cup \{\tilde{h}^{j2}\}_{j=2}^{q}$ and $X^{pq} \stackrel{\text{def}}{=} \{Ux^{i1}\}_{i=1}^{p} \cup \{Ux^{j2}\}_{j=1}^{q}$. Denote by $\Sigma|_{\tilde{H}^{pq}}$ the restriction of Σ to $\tilde{H}^{pq}$, and likewise $\Sigma^{\text{in}}|_{X^{pq}}$ the restriction of Σ^{in} to X^{pq}. We can visualize

$$\Sigma|_{\tilde{H}^{pq}} = \begin{pmatrix} A^{pp} & B^{pq} \\ B^{pq\top} & C^{qq} \end{pmatrix} \in \mathbb{R}^{(p+q-2)\times(p+q-2)} \quad \Sigma^{\text{in}}|_{X^{pq}} = \begin{pmatrix} P^{pp} & Q^{pq} \\ Q^{pq\top} & R^{qq} \end{pmatrix} \in \mathbb{R}^{(p+q)\times(p+q)} \tag{13}$$

where

$$A^{pp} \stackrel{\text{def}}{=} \{\Sigma(\tilde{h}^{i1}, \tilde{h}^{j1})\}_{i,j=2}^{p,p} \qquad C^{qq} \stackrel{\text{def}}{=} \{\Sigma(\tilde{h}^{i2}, \tilde{h}^{j2})\}_{i,j=2}^{q,q} \qquad B^{pq} \stackrel{\text{def}}{=} \{\Sigma(\tilde{h}^{i1}, \tilde{h}^{j2})\}_{i,j=2}^{p,q}$$

$$P^{pp} \stackrel{\text{def}}{=} \{\Sigma^{\text{in}}(Ux^{i1}, Ux^{j1})\}_{i,j=1}^{p,p} \qquad R^{qq} \stackrel{\text{def}}{=} \{\Sigma^{\text{in}}(Ux^{i2}, Ux^{j2})\}_{i,j=1}^{q,q} \qquad Q^{pq} \stackrel{\text{def}}{=} \{\Sigma^{\text{in}}(Ux^{i1}, Ux^{j2})\}_{i,j=1}^{p,q}.$$

Let $\Sigma|_{\tilde{H}^{pq}}^{0}$ also denote $\Sigma|_{\tilde{H}^{pq}}$ padded with an additional column and an additional row of 0s on the left and top of each block $A^{pp}, B^{pq}, B^{pq\top}, C^{qq}$:

$$\Sigma|_{\tilde{H}^{pq}}^{0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & A^{pp} & 0 & B^{pq} \\ 0 & 0 & 0 & 0 \\ 0 & B^{pq\top} & 0 & C^{qq} \end{pmatrix} \in \mathbb{R}^{(p+q)\times(p+q)}. \tag{14}$$

Then Eqs. (11) and (12) can be combined and vectorized as

$$\Sigma|_{\tilde{H}^{p+1,q+1}} = \mathrm{V}_{\phi}\left(\Sigma|_{\tilde{H}^{pq}}^{0} + \Sigma^{\text{in}}|_{X^{pq}} + \sigma_b^2\right). \tag{15}$$

Eq. (15) quickly yields to a iterative, vectorized algorithm for computing $\Sigma|_{\tilde{H}^{tr}}$ (recall t and r are the lengths of the two input sequences), assuming that V_{ϕ} can be efficiently computed (such as those under Definition B.1). Then, with $H^{pq} \stackrel{\text{def}}{=} \{h^{i1}\}_{i=1}^{p} \cup \{h^{j2}\}_{j=1}^{q}$, we have

$$\Sigma|_{H^{tr}} = \Sigma|_{\tilde{H}^{tr}}^{0} + \Sigma^{\text{in}}|_{X^{tr}} + \sigma_b^2.$$

Computing the GP Kernel Finally, given $\Sigma|_{H^{tr}}$, by Corollary 5.5, the covariance of the output of Program 2 in the large n limit is

$$K = \sigma_v^2 \mathrm{V}_\phi\left(\Sigma|_{H^{tr}}\right).$$

Computing RNN kernel

Consider a simple RNN with nonlinearity ϕ, as in Program 2. Suppose we have 2 input sequences $x^{11}, \ldots, x^{t1}$ and $x^{12}, \ldots, x^{r2} \in \mathbb{R}^m$. Assume we sample the RNN parameters as in Eq. (9). Then the outputs of the RNN converge jointly in distribution to a Gaussian with covariance computed as follows.

1. Initialize Σ^{in} according to Eq. (10).
2. Starting with $p = q = 0$, do
   (a) $\Sigma|_{\tilde{H}^{p+1,q+1}} \leftarrow \sigma_w^2 \mathrm{V}_\phi\left(\Sigma|^0_{\tilde{H}^{pq}} + \Sigma^{\text{in}}|_{X^{pq}} + \sigma_b^2\right)$ (see Eqs. (13) and (14) for notations)
   (b) Set $p \leftarrow \min(p+1, t), q \leftarrow \min(q+1, r)$.
   (c) If p and q did not change, break.
3. Compute $\Sigma|_{H^{tr}} \leftarrow \Sigma|^0_{\tilde{H}^{tr}} + \Sigma^{\text{in}}|_{X^{tr}} + \sigma_b^2$.
4. The output kernel is given by

$$K \leftarrow \sigma_v^2 \mathrm{V}_\phi\left(\Sigma|_{H^{tr}}\right).$$

See our repository `github.com/thegregyang/GP4A` for a reference implementation of this kernel.

B.3 Batchnorm (Program 3)

As shown in Appendix A, batchnorm (followed by a coordinatewise nonlinearity) can be just thought of a multivariate nonlinearity, and the computation of Σ can largely follow the same pattern as for any other feedforward neural net (see Program 3). However, doing so efficiently is not so obvious, especially when the batch size is large. In this section, we describe how to overcome this apparent difficulty.

B.3.1 Batchnorm with Single Batch

Let us compute the GP kernel for Program 3, assuming there is only one batch.

Setup The network in Program 3 has parameters $W^1 \in \mathbb{R}^{n\times m}$, where m is the input dimension, and $W^l \in \mathbb{R}^{n\times n}$ for $2 \le l \le L$. Since batchnorm is scale-invariant, we will just assume that

$$W^1_{\alpha\beta} \sim \mathcal{N}(0, 1/m), \quad W^l_{\alpha\beta} \sim \mathcal{N}(0, 1/n), \ \forall l \ge 2,$$

and $v_\alpha \sim \mathcal{N}(0, \sigma_v^2)$. This means that the initial NETSOR sampling data have values

$$\Sigma^{\text{in}}(v, v) = \sigma_v^2$$
$$\Sigma^{\text{in}}(W^1 x^{ia}, W^1 x^{i'a'}) = x^{ia\top} x^{i'a'}/m,$$

and $\mu^{\text{in}} = 0$ identically.

Computing μ As before, from Eq. (2), it is easy to see that $\mu^{\text{in}} = 0$ implies $\mu = 0$.

Computing Σ Applying Eq. (2) in the fashion of Lemma B.5, we get

Lemma B.7. *For any $a \in [k]$ and $2 \le l \le L$, let H^{la} denote the set $\{h^{lia}\}_{i\in[B_a]}$. Also write $H^{1a} \overset{\text{def}}{=} \{W^1 x^{ia}\}_{i\in[B_a]}$. Recall that $\Sigma|_{set}$ denotes the square submatrix of Σ with rows and columns in set. Then for any $a \in [k]$ and $l = 1, \ldots, L-1$,*

$$\Sigma|_{H^{l+1,a}} = \mathop{\mathbb{E}}_{\zeta\sim\mathcal{N}(0,\Sigma|_{H^{la}})} \tilde{\phi}(\zeta)\tilde{\phi}(\zeta)^\top = \mathrm{V}_{\tilde{\phi}}\left(\Sigma|_{H^{la}}\right) \in \mathbb{R}^{B_a\times B_a}, \tag{16}$$

where $\tilde{\phi}$ is batchnorm followed by coordinatewise nonlinearity ϕ, as in Appendix A and Program 3.

A priori, evaluating this expectation requires a B_a-dimensional Gaussian integral, which becomes intractible when B is large. However, if $\phi = \mathrm{ReLU}$, then surprisingly one can reduce the B-dimensional integral this Gaussian expectation seems to require to a 1-dimensional integral: By [65] we can express Eq. (16) as

$$\Sigma|_{H^{l+1,a}} = B_a \int_0^\infty \frac{\mathrm{V}_{\mathrm{ReLU}}(\Sigma^G(I + 2s\Sigma^G)^{-1})}{\sqrt{\det(I + 2s\Sigma^G)}} \,\mathrm{d}s \tag{17}$$

where $\mathrm{V}_{\mathrm{ReLU}}$ is as in Fact B.2, and

$$\begin{aligned} \Sigma^G &\overset{\mathrm{def}}{=} G\Sigma|_{H^{la}}G \\ G &\overset{\mathrm{def}}{=} I_B - \frac{1}{B}\mathbf{1}\mathbf{1}^\top. \end{aligned}$$

B.3.2 Batchnorm with Multiple Batches

Now let's consider the covariance of preactivations between different batches. We maintain the same setup as above, and as usual $\mu = 0$ identically.

Computing Σ By another straightforward application of Eq. (2), we get

Lemma B.8. *With the same notation as in Lemma B.7, for two different batches $a \neq b$,*

$$\Sigma(H^{l+1,a}, H^{l+1,b}) = \mathbb{E}\, \tilde{\phi}(\zeta_1)\tilde{\phi}(\zeta_2)^\top$$

where the expectation is taken over

$$\begin{pmatrix}\zeta_1\\ \zeta_2\end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix}\Sigma(H^{la}, H^{la}) & \Sigma(H^{la}, H^{lb}) \\ \Sigma(H^{lb}, H^{la}) & \Sigma(H^{lb}, H^{lb})\end{pmatrix}\right).$$

Again, this expectaion seems to require an integral in $(B_a + B_b)$ dimensions. However, via some integral tricks, this expectation can be transformed to the following 2D integral:

$$\Sigma(H^{l+1,a}, H^{l+1,b}) = \sqrt{B_a B_b}\pi^{-1} \int_0^\infty \mathrm{d}s \int_0^\infty \mathrm{d}t\; (st)^{-1/2} \det(I_{B_a+B_b} + 2\Omega)^{-1/2} \mathrm{V}_{\mathrm{ReLU}}(\Pi)_{12} \tag{18}$$

where

$$\begin{aligned} \Omega &= D^{1/2}\begin{pmatrix} G_a\Sigma(Y,Y)G_a & G_a\Sigma(Y,Y')G_b \\ G_b\Sigma(Y',Y)G_a & G_b\Sigma(Y',Y')G_b\end{pmatrix} D^{1/2} \\ \Pi &= D^{-1/2}\Omega(I + 2\Omega)^{-1}D^{-1/2} \\ D &= sI_{B_a} \oplus tI_{B_b} = \begin{pmatrix} sI_{B_a} & 0 \\ 0 & tI_{B_b}\end{pmatrix} \\ G_a &= I_{B_a} - B_a^{-1}\mathbf{1}\mathbf{1}^\top \\ G_b &= I_{B_b} - B_b^{-1}\mathbf{1}\mathbf{1}^\top \end{aligned}$$

and $\mathrm{V}_{\mathrm{ReLU}}(\Pi)_{12}$ is the block of $\mathrm{V}_{\mathrm{ReLU}}(\Pi)$ on the first row, second column, of size $B_a \times B_b$.

Computing the GP Kernel K The computation is similar to Lemmas B.7 and B.8, so let us directly summarize the entire algorithm below.

Computing Batchnorm+ReLU kernel

We compute the kernel of a L-layer fully-connected network where each layer has a batchnorm followed by ReLU, as shown in Program 3. Let the input dimension be m and the common width of hidden layers be n. Sample the first layer weights W^1 as $W^1_{\alpha\beta} \sim \mathcal{N}(0, \sigma_W^2/m)$ and each higher layer's weight matrix W^l as $W^l_{\alpha\beta} \sim \mathcal{N}(0, \sigma_W^2/n)$ with $\sigma_W = 1$ (since batchnorm is scale-invariant), and the readout layer as $v_\alpha \sim \mathcal{N}(0, \sigma_v^2)$. We omit biases since batchnorm is shift invariant. Suppose we have k batches of inputs $\{x^{ib} : i \in [B_b], b \in [k]\}$, with batch b containing B_b inputs. Then the outputs of the network converge jointly in distribution to a Gaussian $\mathcal{N}(0, K)$ with K computed as follows.

```
1. Initialize $\{K^0_{ab} \in \mathbb{R}^{B_a \times B_b}\}_{a,b=1}^k$ by $(K^0_{ab})_{ij} \leftarrow x^{ia\top} x^{jb}/m$.
2. For $l = 1, \ldots, L$, do
   (a) For $a = 1, \ldots, k$, do
       i. $K^l_{aa} \leftarrow \mathrm{V}_{\tilde{\phi}}(K^{l-1}_{aa})$ by evaluating a 1D integral according to Eq. (17).
   (b) For $a, b \in [k]$, $a \neq b$, do
       i. Compute $K^l_{ab}$ by using $K^{l-1}_{ab}, K^{l-1}_{aa}, K^{l-1}_{bb}$ and evaluating a 2D integral
          according to Eq. (18).
3. Return
```

$$K \leftarrow \sigma_v^2 \begin{pmatrix} K^L_{11} & \cdots & K^L_{1k} \\ \vdots & \ddots & \vdots \\ K^L_{k1} & \cdots & K^L_{kk}. \end{pmatrix} \in \mathbb{R}^{\sum_a B_a \times \sum_a B_a}$$

Vectorized Implementation In our repo github.com/thegregyang/GP4A, we show how to implement single- and multi-batch BN using the quadpy [48] package for vectorized quadrature integration, and by using eigendecomposition to simplify the computation of the integrand in the integrals above.

B.4 Convolution and Pooling (Program 4)

Convolution and pooling, in the context of neural network-Gaussian process correspondence, have already been treated in [43, 18]. In this section we revisit the same derivations from the perspective of tensor programs.

B.4.1 CNN with Single Input

Let us compute the GP kernel for Program 4, for the following setup:

Setup The CNN in Program 4 has parameters $\{W^1_j\}_{j \in ker^1}, \{W^2_j\}_{j \in ker^2}$. It has widths n^1 and n^2, but for simplicity, let's assume $n^1 = n^2 = n$. Each input image is given as $\{x_i \in \mathbb{R}^m\}_{i \in pos^0}$ where pos^0 denotes "pixel locations" and m denotes number of channels (for example, $pos^0 = [32] \times [32]$ and $m = 3$ for the CIFAR10 dataset). Suppose we sample the parameters as

$$(W^1_j)_{\alpha\beta} \sim \mathcal{N}(0, \sigma_w^2/m),\ \forall j \in ker^1, \quad (W^2_j)_{\alpha\beta} \sim \mathcal{N}(0, \sigma_w^2/n^1),\ \forall j \in ker^2, \quad v_\alpha \sim \mathcal{N}(0, \sigma_v^2/n).$$

This induces Σ^{in} as follows:

$$\Sigma^{\text{in}}(v, v) = \sigma_v^2$$
$$\Sigma^{\text{in}}(W^1_j x_{i+j}, W^1_{j'} x_{i'+j'}) = \sigma_w^2 x_{i+j}^\top x_{i'+j'}/m$$

for any $i, i' \in pos^1, j, j' \in ker^1$ such that $i + j, i' + j' \in pos^0$; and $\Sigma^{\text{in}}(g, g') = 0$ for any other pairs of input G-vars. In addition, $\mu^{\text{in}} = 0$ identically.

Computing μ As before, from Eq. (2), it is easy to see that $\mu^{\text{in}} = 0$ implies $\mu = 0$.

Computing Σ By straightforward applications of Eq. (2), we obtain the following

Lemma B.9. *For any $i, i' \in pos^1$,*

$$\Sigma(h_i^1, h_{i'}^1) = \sum_{j,j' \in ker^1} \Sigma^{\text{in}}(W_j^1 x_{i+j}, W_{j'}^1 x_{i'+j'}).$$

In addition, for any $j, j' \in ker^2$ such that $i + j, i' + j' \in pos^1$,

$$\Sigma(h_{i;j}^2, h_{i';j'}^2) = \sigma_w^2 \mathop{\mathbb{E}}_{z_1,z_2} \phi(z_1)\phi(z_2)\mathbb{I}(j = j'),$$

where $(z_1, z_2) \sim \mathcal{N}(0, \Sigma|_{h_i^1, h_{i'}^1})$. Finally, for any $i, i' \in pos^2$,

$$\Sigma(h_i^2, h_{i'}^2) = \sum_{j,j' \in ker^2} \Sigma(h_{i;j}^2, h_{i';j'}^2)$$

where the sum is over all $j, j' \in ker^2$ such that $i + j, i' + j' \in pos^1$.

Computing the GP kernel K By Corollary 5.5, the output of the CNN converges in distribution to $\mathcal{N}(0, K)$ where K is a scalar given by

$$K \stackrel{\text{def}}{=} \lim_{n\to\infty} \operatorname{Var}\left(\frac{v^\top \bar{x}^2}{n}\right) = \sigma_v^2 \mathop{\mathbb{E}}_{Z} \left(\frac{1}{|pos^2|} \sum_i \phi(Z^{h_i^2})\right)^2 = \sigma_v^2 \mathop{\mathbb{E}}_{i,i' \in pos^2} \mathop{\mathbb{E}}_{Z} \phi(Z^{h_i^2})\phi(Z^{h_{i'}^2})$$

where $Z \sim \mathcal{N}(\mu, \Sigma)$ and where in the last expression, i, i' are sampled independently and uniformly from pos^2. If we let $H^2 \stackrel{\text{def}}{=} \{h_i^2\}_{i \in pos^2}$, then K can be computed as

$$K = \sigma_v^2 \mathop{\mathbb{E}}_{i,i' \in pos^2} \Lambda_{ii'} \quad \text{where} \quad \Lambda \stackrel{\text{def}}{=} \mathrm{V}_\phi\left(\Sigma|_{H^2}\right).$$

B.4.2 CNN with Multiple Inputs

Now consider the general case when we have multiple inputs $x^1, \ldots, x^B$ and have L layers (but, for simplicity, still no bias), as in Program 7. The derivation is very similar to the single input case, but we will err on the side of completeness.

Setup The CNN in Program 7 has parameters $\{W^l\}_{l \in [L], j \in ker^l}$. It has widths $n^1, \ldots, n^L$, but as before, we shall assume they are all equal to n for simplicity. Each input image x^a is given as $\{x_i^a \in \mathbb{R}^m\}_{i \in pos^0}$ where pos^0 denotes "pixel locations" and m denotes number of channels. Suppose we sample the parameters as

$$(W_j^1)_{\alpha\beta} \sim \mathcal{N}(0, \sigma_w^2/m),\ \forall j \in ker^1, \quad (W_j^l)_{\alpha\beta} \sim \mathcal{N}(0, \sigma_w^2/n^{l-1}),\ \forall j \in ker^l,$$

for all $l = 2, \ldots, L$, and $v_\alpha \sim \mathcal{N}(0, \sigma_v^2/n)$. This induces Σ^{in} as follows:

$$\Sigma^{\text{in}}(v, v) = \sigma_v^2$$
$$\Sigma^{\text{in}}(W_j^1 x_{i+j}^a, W_{j'}^1 x_{i'+j'}^{a'}) = \sigma_w^2 x_{i+j}^{a}{}^\top x_{i'+j'}^{a'}/m$$

for any $a, a' \in [B]$ and any $i, i' \in pos^1, j, j' \in ker^1$ such that $i+j, i'+j' \in pos^0$; and $\Sigma^{\text{in}}(g, g') = 0$ for any other pairs of input G-vars. In addition, $\mu^{\text{in}} = 0$ identically.

Computing μ As before, from Eq. (2), it is easy to see that $\mu^{\text{in}} = 0$ implies $\mu = 0$.

Computing Σ By straightforward applications of Eq. (2), we obtain the following

Lemma B.10. *For any $a, a' \in [B]$ and any $i, i' \in pos^1$,*

$$\Sigma(h_i^{1a}, h_{i'}^{1a'}) = \sum_{j,j' \in ker^1} \Sigma^{\text{in}}(W_j^1 x_{i+j}^a, W_{j'}^1 x_{i'+j'}^{a'}). \tag{19}$$

In addition, for any $2 \leq l \leq L$ and any $j, j' \in ker^2$ such that $i + j, i' + j' \in pos^1$,

$$\Sigma(h_{i;j}^{la}, h_{i';j'}^{la'}) = \sigma_w^2 \mathop{\mathbb{E}}_{z_1,z_2} \phi(z_1)\phi(z_2)\mathbb{I}(j = j'), \tag{20}$$

NETSOR program 7 L-layer Convolutional Network with Global Average Pooling

```
Input: ${W^1_j x^a_{i+j} : \mathsf{G}(n^1)}_{a\in[B],\ j\in ker^1, i\in pos^1\ s.t.\ i+j\in pos^0}$
Input: ${W^l_j : \mathsf{A}(n^l, n^{l-1})}_{2\le l\le L, j\in ker^l}$
  // Readout weights
Input: $v : \mathsf{G}(n^L)$
  for $a \in [B]$ do
      // Layer 1 convolution
      for $i \in pos^1$ do
          // Directly use input embeddings
          // LinComb
          // Sum is over all $j \in ker^1$ such that
          // there is $i' \in pos^0$ with $i' = j + i$
          $h^{1a}_i := \sum_j W^1_j x^a_{i+j} : \mathsf{G}(n^1)$
      end for
      // Higher layer convolutions
      for $l = 2, \dots, L$ do
          for $i \in pos^{l-1}$ do
              $x^{l-1,a}_i := \phi(h^{l-1,a}_i) : \mathsf{H}(n^{l-1})$
          end for
          for $j \in ker^l, i \in pos^l\ s.t.\ i + j \in pos^{l-1}$ do
              // MatMul
              $h^{la}_{i;j} := W^l_j x^{l-1,a}_{i+j} : \mathsf{G}(n^l)$
          end for
          for $i \in pos^l$ do
              // Sum is over all $j \in ker^l$ such that
              // there is $i' \in pos^{l-1}$ with $i' = j + i$
              $h^{la}_i := \sum_j h^{la}_{i;j} : \mathsf{G}(n^l)$
          end for
      end for
      // Nonlinearity & Global Average Pooling
      $\bar{x}^{La} := \frac{1}{|pos^L|} \sum_{i\in ker^L} \phi(h^{La}_i) : \mathsf{H}(n^L)$
  end for
Output: $v^\top \bar{x}^{L1}/\sqrt{n^L}, \dots, v^\top \bar{x}^{LB}/\sqrt{n^L}$
```

where $(z_1, z_2) \sim \mathcal{N}\left(0, \Sigma|_{h^{l-1,a}_i, h^{l-1,a'}_{i'}}\right)$. *Finally, for any* $i, i' \in pos^l$,

$$\Sigma(h^{la}_i, h^{la'}_{i'}) = \sum_{j,j'} \Sigma(h^{la}_{i;j}, h^{la'}_{i';j'}) \tag{21}$$

where the sum is over all $j, j' \in ker^l$ *such that* $i + j, i' + j' \in pos^{l-1}$.

These equations are all we need to compute the GP kernel K.

Computing the GP kernel K By Corollary 5.5, the output of the CNN converges in distribution to $\mathcal{N}(0, K)$ where $K \in \mathbb{R}^{B\times B}$ is given by

$$K_{aa'} \overset{\text{def}}{=} \lim_{n\to\infty} \mathrm{Cov}\left(\frac{v^\top \bar{x}^{La}}{\sqrt{n}}, \frac{v^\top \bar{x}^{La'}}{\sqrt{n}}\right) = \sigma_v^2 \mathop{\mathbb{E}}_Z \left(\frac{1}{|pos^L|}\sum_i \phi(Z^{h^{La}_i})\right)\left(\frac{1}{|pos^L|}\sum_i \phi(Z^{h^{La'}_i})\right)$$
$$= \sigma_v^2 \mathop{\mathbb{E}}_{i,i'\in pos^L} \mathop{\mathbb{E}}_Z \phi(Z^{h^{La}_i})\phi(Z^{h^{La'}_{i'}}) \tag{22}$$

where $Z \sim \mathcal{N}(\mu, \Sigma)$ and where in the last expression, i, i' are sampled independently and uniformly from pos^L. Since $\Sigma(h^{La}_i, h^{La'}_{i'})$ can be obtained recursively via Lemma B.10, one can compute K easily via recursion. But we can do better by vectorizing the whole computation.

Vectorized Implementation Let us define the 4-tensor

$$\mathbf{\Sigma}^l = \{\mathbf{\Sigma}^l_{aa'ii'} : a, a' \in [B], i, i' \in pos^l\}$$

by

$$\mathbf{\Sigma}^l_{aa'ii'} \overset{\text{def}}{=} \Sigma(h^{la}_i, h^{la'}_{i'}).$$

Then Eq. (19) corresponds to

$$\mathbf{\Sigma}^1_{aa'ii'} = \sigma_w^2 \sum_{j,j'} {x^a_{i+j}}^\top x^{a'}_{i'+j'}/m$$

where the sum is over all $j, j' \in ker^1$ such that $i + j, i' + j' \in pos^0$. For $l = 2, \dots, L-1$, Eqs. (20) and (21) can be vectorized as

$$\mathbf{\Sigma}^l_{aa'} = \kappa^l * \hat{\mathbf{\Sigma}}^l_{aa'}, \quad \text{where} \quad \hat{\mathbf{\Sigma}}^l = \sigma_w^2 \mathrm{V}_\phi(\mathbf{\Sigma}^{l-1}),$$

treating $\mathbf{\Sigma}^{l-1}$ as a $(B \cdot pos^{l-1}) \times (B \cdot pos^{l-1})$ matrix, and $\kappa^l *$ is the "convolution"

$$(\kappa^l * \hat{\mathbf{\Sigma}}^l_{aa'})_{ii'} = \sum_{\substack{j,j' \in ker^l \\ i+j \in pos^{l-1} \\ i'+j' \in pos^{l-1}}} (\hat{\mathbf{\Sigma}}^l_{aa'})_{i+j, i'+j'}. \tag{23}$$

This κ^l convolution can indeed be implemented as a (CUDA) convolutional operation, vectorized over all a, a'.

Finally, to obtain the infinite-width GP kernel K of the CN output, we can vectorize Eq. (22) as

$$K_{aa'} = \sigma_v^2 E * \hat{\mathbf{\Sigma}}^L, \quad \text{where} \quad \hat{\mathbf{\Sigma}}^L = \mathrm{V}_\phi(\mathbf{\Sigma}^{L-1}),$$

and $E*$ denotes the spacial averaging

$$(E * \hat{\mathbf{\Sigma}}^L)_{aa'} \overset{\text{def}}{=} \mathop{\mathbb{E}}_{i,i' \in pos^L} \hat{\mathbf{\Sigma}}^L_{aa'ii'}. \tag{24}$$

Again, $E*$ can be implemented as a convolution operator vectorized over all a, a'.

In summary,

Computing CNN Kernel

Suppose we have an L-layer convolutional neural network with coordinatewise nonlinearity ϕ but no bias, as in Program 7, that takes images with m channels and of size $pos^0 \times pos^0$. Suppose we have a set of inputs $x^1, \dots, x^B$ where each input x^a is given as $x^a = \{x^a_i \in \mathbb{R}^m\}_{i \in pos^0}$. Then the CNN outputs converge in distribution to a Gaussian $\mathcal{N}(0, K)$ where $K \in \mathbb{R}^{B \times B}$ can be calculated as follows.

1. Initialize $\mathbf{\Sigma}^1 \in \mathbb{R}^{B \times B \times pos^l \times pos^l}$ by

$$\mathbf{\Sigma}^1_{aa'ii'} = \sigma_w^2 \sum_{j,j'} {x^a_{i+j}}^\top x^{a'}_{i'+j'}/m$$

   where the sum is over all $j, j' \in ker^1$ such that $i + j, i' + j' \in pos^0$.
2. For $l = 2, \dots, L-1$, do
   (a) $\mathbf{\Sigma}^l \leftarrow \sigma_w^2 \kappa^l * \mathrm{V}_\phi(\mathbf{\Sigma}^{l-1})$ (see Eq. (23) for κ^l's definition)
3. return $K \leftarrow \sigma_v^2 E * \mathrm{V}_\phi\left(\mathbf{\Sigma}^{L-1}\right)$ (see Eq. (24) for E's definition)

B.5 GRU (Program 5)

We demonstrate how to compute the GP kernel for GRU as encoded in NETSOR by Program 5. A key distinguishing feature of this conversion is that we will need to compute high dimensional Gaussian expectations, where the dimension is as large as the number of timesteps unrolled, in contrast to the simple RNN case Program 2. These Gaussian expectations correspond to the expected values of multiplication of the gate values across time.

We first proceed with a single input sequence, as in Program 5. We then comment on the generalization to multiple sequences at the end.

Setup We will obtain the μ and Σ for Program 5 with

$$\begin{aligned}(U_{\mathsf{z}})_{\alpha\beta}, (U_{\mathsf{r}})_{\alpha\beta}, (U_{\mathsf{h}})_{\alpha\beta} &\sim \mathcal{N}(0, \sigma_U^2/n), & (W_{\mathsf{z}})_{\alpha\beta}, (W_{\mathsf{r}})_{\alpha\beta}, (W_{\mathsf{h}})_{\alpha\beta} &\sim \mathcal{N}(0, \sigma_W^2/n), \\ (b_{\mathsf{z}})_\alpha, (b_{\mathsf{r}})_\alpha, (b_{\mathsf{h}})_\alpha &\sim \mathcal{N}(0, \sigma_b^2), & v_\alpha &\sim \mathcal{N}(0, \sigma_v^2), \text{ and } h^0 = 0.\end{aligned} \tag{25}$$

Suppose the input tokens x^i to the GRU have dimension m. The randomization of U induces the following covariance structure in the input token embeddings

$$\Sigma^{\text{in}}(Ux, Uy) = \sigma_U^2 x^\top y/m$$

for any $x, y \in \{x^i\}_{i=1}^T$. For any other pair g, g' of input G-vars, $\Sigma^{\text{in}}(g, g') = 0$. Additionally, $\mu^{\text{in}}(g) = 0$ for all input G-vars g.

Computing μ In fact, one can quickly see that $\mu(g) = 0$ for *all* G-vars g.

Computing Σ Applying Eq. (2) to Program 5 and some simplification in the manner of Lemma B.6's proof yields, for any two times t, s,

$$\Sigma(\tilde{z}^t, \tilde{z}^s) = \Sigma(h_{\mathsf{z}}^t, h_{\mathsf{z}}^s) + \sigma_U^2 x^{t\top} x^s/m + \sigma_b^2 \tag{26}$$

$$\Sigma(\tilde{r}^t, \tilde{r}^s) = \Sigma(h_{\mathsf{r}}^t, h_{\mathsf{r}}^s) + \sigma_U^2 x^{t\top} x^s/m + \sigma_b^2 \tag{27}$$

$$\Sigma(\tilde{h}^t, \tilde{h}^s) = \Sigma(h_{\mathsf{h}}^t, h_{\mathsf{h}}^s) + \sigma_U^2 x^{t\top} x^s/m + \sigma_b^2 \tag{28}$$

$$\begin{aligned}\Sigma(h_{\mathsf{z}}^t, h_{\mathsf{z}}^s) &= \Sigma(h_{\mathsf{r}}^t, h_{\mathsf{r}}^s) \\ &= \sigma_W^2 \sum_{i=1}^t \sum_{j=1}^s \Bigg\{ \mathbb{E}\, \phi(Z^{\tilde{h}^i}) \phi(Z^{\tilde{h}^j}) \\ &\qquad \times \mathbb{E}\left[\sigma(Z^{\tilde{z}^i}) \prod_{p=i+1}^t (1 - \sigma(Z^{\tilde{z}^p}))\right] \times \left[\sigma(Z^{\tilde{z}^j}) \prod_{q=j+1}^s (1 - \sigma(Z^{\tilde{z}^q}))\right] \Bigg\}\end{aligned} \tag{29}$$

$$\Sigma(h_{\mathsf{h}}^t, h_{\mathsf{h}}^s) = \sigma_W^2 \Sigma(h_{\mathsf{z}}^t, h_{\mathsf{z}}^s)\, \mathbb{E}\, \sigma(Z^{\tilde{r}^t}) \sigma(Z^{\tilde{r}^s}) \tag{30}$$

where expectations are taken over $Z = \{Z^g\}_{g \text{ is G-var}} \sim \mathcal{N}(\mu, \Sigma)$, which has one component for each G-var in the program. Then, applying Corollary 5.5, we see that the output of the GRU

$$(v^\top h^1/\sqrt{n}, \ldots, v^\top h^T/\sqrt{n})$$

converges in distribution to a zero mean Gaussian distribution with covariance matrix $K = \{K_{ts}\}_{t,s=1}^T$,

$$\begin{aligned}K_{ts} = \sigma_v^2 \sum_{i=1}^t \sum_{j=1}^s \Bigg\{ &\mathbb{E}\, \phi(Z^{\tilde{h}^i}) \phi(Z^{\tilde{h}^j}) \\ &\times \mathbb{E}\left[\sigma(Z^{\tilde{z}^i}) \prod_{p=i+1}^t (1 - \sigma(Z^{\tilde{z}^p}))\right] \times \left[\sigma(Z^{\tilde{z}^j}) \prod_{q=j+1}^s (1 - \sigma(Z^{\tilde{z}^q}))\right] \Bigg\}.\end{aligned} \tag{31}$$

Eqs. (27) to (31) yield the complete set of equations to compute the output covariance K, but to do so efficiently rests entirely on evaluating the the possibly T-dimensional integral behind

$$\mathbb{E}\left[\sigma(Z^{\tilde{z}^i}) \prod_{p=i+1}^t (1 - \sigma(Z^{\tilde{z}^p}))\right] \times \left[\sigma(Z^{\tilde{z}^j}) \prod_{q=j+1}^s (1 - \sigma(Z^{\tilde{z}^q}))\right]. \tag{32}$$

For general σ and ϕ, this is hopeless. However, when $\phi = \text{erf}$ and $\sigma = (1 + \text{erf})/2$ — which approximate $\phi = \tanh$ and $\sigma = \text{sigmoid}$ — Eq. (32) can in fact be evaluated efficiently by reducing it to a Gaussian orthant probability, which can be evaluated efficiently [19]:

NETSOR program 8 GRU, Multiple Input Sequences

```
  // Embeddings of $B$ input sequences
  // with $a$th sequence having length $T_a$
Input: $\{U_z x^{ta} : \mathsf{G}(n)\}_{a\in[B],t\in[T_a]}$
Input: $\{U_r x^{ta} : \mathsf{G}(n)\}_{a\in[B],t\in[T_a]}$
Input: $\{U_h x^{ta} : \mathsf{G}(n)\}_{a\in[B],t\in[T_a]}$
  // Parameters
Input: $W_z, W_r, W_h : \mathsf{A}(n,n)$
Input: $b_z, b_r, b_h : \mathsf{G}(n)$
  // Initial GRU state
Input: $h^0 : \mathsf{G}(n)$
  // Readout layer
Input: $v : \mathsf{G}(n)$
  for $a \in [B]$ do
      for $t \in [T_a]$ do
          $h_z^{ta} := W_z h^{t-1,a} : \mathsf{G}(n)$
          $\tilde{z}^{ta} := h_z^{ta} + U_z x^{ta} + b_z : \mathsf{G}(n)$
          $h_r^{ta} := W_r h^{t-1,a} : \mathsf{G}(n)$
          $\tilde{r}^{ta} := h_r^{ta} + U_r x^{ta} + b_r : \mathsf{G}(n)$
          // Morally, $\hat{h}^{t-1,a} = \sigma(\tilde{r}^{t-1,a}) \odot h^{t-1,a}$, but we need to unroll $h^{t-1,a}$ to apply Nonlin
          $\hat{h}^{t-1,a} := \sigma(\tilde{r}^{t-1,a}) \odot$
              $\left(h^0 \odot \bigodot_{i=1}^{t-1}(1-\sigma(\tilde{z}^{ia})) + \sum_{j=1}^{t-1} \phi(\tilde{h}^{ja}) \odot \sigma(\tilde{z}^{ja}) \odot \bigodot_{l=j+1}^{t-1}(1-\sigma(\tilde{z}^{la}))\right) : \mathsf{H}(n)$
          $h_h^{ta} := W_h \hat{h}^{t-1,a} : \mathsf{G}(n)$
          $\tilde{h}^{ta} := h_h^{ta} + U_h x^{ta} + b_h : \mathsf{G}(n)$
          // Unrolling $h^t$ to a coordinatewise function of G-vars
          $h^t := h^0 \odot \bigodot_{i=1}^{t}(1-\sigma(\tilde{z}^{ia})) + \sum_{j=1}^{t} \phi(\tilde{h}^{ja}) \odot \sigma(\tilde{z}^{ja}) \odot \bigodot_{l=j+1}^{t}(1-\sigma(\tilde{z}^{la})) : \mathsf{H}(n)$
      end for
  end for
Output: $\{v^\top h^{ta}/\sqrt{n}\}_{a\in[B],t\in[T_a]}$
```

Lemma B.11. *Let $\phi = \mathrm{erf}$ and $\sigma = (1+\mathrm{erf})/2$. Then for any $\mu \in \mathbb{R}^T$ and any PSD $\Sigma \in \mathbb{R}^{T\times T}$,*

$$\begin{aligned}
\mathop{\mathbb{E}}_{x\sim\mathcal{N}(\mu,\Sigma)} \prod_{i=1}^T \phi(x_i) &= \mathop{\mathbb{E}}_{x\sim\mathcal{N}(\mu,\Sigma+\frac{1}{2}I)} \prod_{i=1}^T \mathrm{sgn}(x_i) \\
\mathop{\mathbb{E}}_{x\sim\mathcal{N}(\mu,\Sigma)} \prod_{i=1}^T \sigma(x_i) &= \mathop{\mathbb{E}}_{x\sim\mathcal{N}(\mu,\Sigma+\frac{1}{2}I)} \prod_{i=1}^T \mathbb{I}(x_i \geq 0) \\
&= \Pr_{x\sim\mathcal{N}(\mu,\Sigma+\frac{1}{2}I)}[x \geq 0].
\end{aligned}$$

Remark B.12. Observe that if $T = 2$, then Lemma B.11 recovers the arccosine kernel of erf via Fact B.3:

$$\mathbb{E}\left[\mathrm{erf}(x_1)\mathrm{erf}(x_2) : x \sim \mathcal{N}(0,\Sigma)\right] = \frac{2}{\pi}\arcsin\frac{\Sigma_{12}}{\sqrt{(\Sigma_{11}+\frac{1}{2})(\Sigma_{22}+\frac{1}{2})}}.$$

To apply Lemma B.11, we can express Eq. (32) as

$$\begin{aligned}
&\mathbb{E}\left[\sigma(Z^{\tilde{z}^i})\prod_{p=i+1}^{t}\sigma(-Z^{\tilde{z}^p})\right]\times\left[\sigma(Z^{\tilde{z}^j})\prod_{q=j+1}^{s}\sigma(-Z^{\tilde{z}^q})\right]\\
&=\mathbb{E}\left[\sigma(\hat{Y}^i)\prod_{p=i+1}^{s}\sigma(\hat{Y}^p)\right]\times\left[\sigma(\check{Y}^j)\prod_{q=j+1}^{s}\sigma(\check{Y}^q)\right]\\
&=\mathbb{E}\left[\prod_{p=i}^{t}\sigma(\hat{Y}^p)\right]\times\left[\prod_{q=j}^{s}\sigma(\check{Y}^q)\right]
\end{aligned}$$

where $(\hat{Y}^i,\ldots,\hat{Y}^t,\check{Y}^j,\ldots,\check{Y}^s)\sim\mathcal{N}(\nu,\Omega)$ with b and Ω given as follows

$$\begin{aligned}
\nu(\hat{Y}^i)&=\mu(\tilde{z}^i) & \nu(\check{Y}^j)&=\mu(\tilde{z}^j)\\
\nu(\hat{Y}^p)&=-\mu(\tilde{z}^p),\forall p\geq i+1 & \nu(\check{Y}^q)&=-\mu(\tilde{z}^q),\forall q\geq j+1
\end{aligned}$$

$$\begin{aligned}
\Omega(\hat{Y}^p,\check{Y}^q)&=\begin{cases}\Sigma(\tilde{z}^p,\tilde{z}^q) & \text{if } p\geq i+1, q\geq j+1, \text{ or } p=i, q=j\\ -\Sigma(\tilde{z}^p,\tilde{z}^q) & \text{otherwise.}\end{cases}\\
\Omega(\hat{Y}^p,\hat{Y}^{p'})&=\begin{cases}\Sigma(\tilde{z}^p,\tilde{z}^{p'}) & \text{if } p,p'\geq i+1, \text{ or } p=p'=i\\ -\Sigma(\tilde{z}^p,\tilde{z}^{p'}) & \text{otherwise.}\end{cases}\\
\Omega(\check{Y}^q,\check{Y}^{q'})&=\begin{cases}\Sigma(\tilde{z}^q,\tilde{z}^{q'}) & \text{if } q,q'\geq j+1, \text{ or } q=q'=j\\ -\Sigma(\tilde{z}^q,\tilde{z}^{q'}) & \text{otherwise.}\end{cases}
\end{aligned}$$

Using Lemma B.11, one then has

$$\begin{aligned}
&\mathbb{E}\left[\sigma(Z^{\tilde{z}^i})\prod_{p=i+1}^{t}\sigma(-Z^{\tilde{z}^p})\right]\times\left[\sigma(Z^{\tilde{z}^j})\prod_{q=j+1}^{s}\sigma(-Z^{\tilde{z}^q})\right]\\
&=\Pr\left[X\geq 0: X\sim\mathcal{N}\left(\nu,\frac{1}{2}I+\Omega\right)\right]. \qquad (33)
\end{aligned}$$

If we two input sequences, the equations for recursively computing Σ and of K are similar to the above, and we summarize them below

Computing the GRU kernel

Consider a GRU processing B sequences in the fashion of Program 8, with $\phi = \mathrm{erf}$ and $\sigma = (1+\mathrm{erf})/2$. Sample the GRU's parameters as in Eq. (25). Then for sequence numbers $a, b \in [B]$ and time steps $2 \le t \le T_a, 2 \le s \le T_b$, we have the following recurrence relations

$$\begin{aligned}
\Sigma(\tilde{z}^{ta}, \tilde{z}^{sb}) &= \Sigma(h_{\mathsf{z}}^{ta}, h_{\mathsf{z}}^{sb}) + \sigma_U^2 x^{ta\top} x^{sb}/m + \sigma_b^2 \\
\Sigma(\tilde{r}^{ta}, \tilde{r}^{sb}) &= \Sigma(h_{\mathsf{r}}^{ta}, h_{\mathsf{r}}^{sb}) + \sigma_U^2 x^{ta\top} x^{sb}/m + \sigma_b^2 \\
\Sigma(\tilde{h}^{ta}, \tilde{h}^{sb}) &= \Sigma(h_{\mathsf{h}}^{ta}, h_{\mathsf{h}}^{sb}) + \sigma_U^2 x^{ta\top} x^{sb}/m + \sigma_b^2 \\
\Sigma(h_{\mathsf{z}}^{ta}, h_{\mathsf{z}}^{sb}) &= \Sigma(h_{\mathsf{r}}^{ta}, h_{\mathsf{r}}^{sb}) \\
&= \sigma_W^2 \sum_{i=1}^{t} \sum_{j=1}^{s} \zeta_{i:t,j:s}^{ab} \,\mathbb{E}\, \phi(Z^{\tilde{h}^{ia}}) \phi(Z^{\tilde{h}^{jb}}) \\
\Sigma(h_{\mathsf{h}}^{ta}, h_{\mathsf{h}}^{sb}) &= \sigma_W^2 \Sigma(h_{\mathsf{z}}^{ta}, h_{\mathsf{z}}^{sb}) \,\mathbb{E}\, \sigma(Z^{\tilde{r}^{ta}}) \sigma(Z^{\tilde{r}^{sb}})
\end{aligned}$$

with initial conditions

$$\Sigma(\tilde{z}^{ta}, \tilde{z}^{sb}) = \Sigma(\tilde{r}^{ta}, \tilde{r}^{sb}) = \Sigma(\tilde{h}^{ta}, \tilde{h}^{sb}) = 0 \qquad \text{if } t = 1 \text{ or } s = 1,$$

and the output covariance K of the GRU outputs in the large n limit can be computed as

$$\begin{aligned}
K_{ta,sb} &= \lim_{n\to\infty} \mathrm{Cov}\left(\frac{v^\top x^{ta}}{\sqrt{n}}, \frac{v^\top x^{sb}}{\sqrt{n}} \right) \\
&= \sigma_v^2 \sum_{i=1}^{t} \sum_{j=1}^{s} \zeta_{i:t,j:s}^{ab} \,\mathbb{E}\, \phi(Z^{\tilde{h}^{ia}}) \phi(Z^{\tilde{h}^{jb}})
\end{aligned}$$

where $Z \sim \mathcal{N}(0, \Sigma)$ and

$$\zeta_{i:t,j:s}^{ab} \overset{\text{def}}{=} \mathop{\mathbb{E}}_{Z} \left[\sigma(Z^{\tilde{z}^{ia}}) \prod_{p=i+1}^{t} (1 - \sigma(Z^{\tilde{z}^{pa}})) \right] \times \left[\sigma(Z^{\tilde{z}^{jb}}) \prod_{q=j+1}^{s} (1 - \sigma(Z^{\tilde{z}^{qb}})) \right],$$

which can be reduced to a computation of orthant probability in the fashion of Eq. (33).

The above equations can be turned into a (relatively) efficient algorithm for computing the GP kernel of a GRU. Our repo `github.com/thegregyang/GP4A` shows a reference implementation of it (allowing slightly more general initialization hyperparameters). It leverages the `R` package `mvtnorm` [19] to evaluate the Gaussian orthant probability involved in Eq. (33).

In the rest of the section, we prove Lemma B.11.

Review of (Tempered) Distributions Before we begin the proof of Lemma B.11, we briefly recall the notion of a tempered distribution, which is a "pseudo-function" that is formally defined as an element of the dual of Schwartz space (intuitively, the space of functions with rapidly decreasing derivatives of all orders) [52]. Given a Schwartz function f and a tempered distribution τ, the value of τ on f will be denoted here by

$$\langle \tau, f \rangle.$$

For example, if τ is a locally-integrable function, then τ is also a tempered distribution and $\langle \tau, f \rangle$ can be defined by

$$\langle \tau, f \rangle = \int \tau(x) f(x) \,\mathrm{d}x.$$

As all Schwartz functions have Fourier transforms [52], any tempered distribution has Fourier transform defined by

$$\langle \mathcal{F}\{\tau\}, f \rangle \overset{\text{def}}{=} \langle \tau, \mathcal{F}\{f\} \rangle.$$

In what follows, notationally, Fourier transform will convert functions or distributions in variable t to functions to distributions in variable x, or vice versa. See [52] for more background on distributions.

Proof of Lemma B.11. As a tempered distribution, $\phi = \operatorname{erf}$ can be expressed as

$$\begin{aligned}\phi(x) &= \mathcal{F}\left\{\frac{-i\sqrt{2}}{\sqrt{\pi}}\mathrm{p.v.}\frac{e^{-t^2/4}}{t}\right\}(x)\\ &= \frac{1}{\sqrt{2\pi}}\mathrm{p.v.}\int e^{ixt}\frac{-i\sqrt{2}}{\sqrt{\pi}}\frac{e^{-t^2/4}}{t}\,\mathrm{d}t,\end{aligned}$$

where $\mathrm{p.v.}$ denotes principal value integration

$$\mathrm{p.v.}\int e^{ixt}\frac{-i\sqrt{2}}{\sqrt{\pi}}\frac{e^{-t^2/4}}{t}\,\mathrm{d}t \overset{\text{def}}{=} \lim_{\varepsilon\to 0}\left(\int_{-\infty}^{-\varepsilon}+\int_{\varepsilon}^{\infty}\right)e^{ixt}\frac{-i\sqrt{2}}{\sqrt{\pi}}\frac{e^{-t^2/4}}{t}\,\mathrm{d}t.$$

Over multiple variables, because Fourier transform over $\mathbb{R}^T$ is equivalent to applying 1D Fourier transform over each coordinate, we have

$$\prod_{i=1}^T \phi(x_i) = \mathcal{F}\left\{\left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\frac{e^{-\sum_{i=1}^T t_i^2/4}}{\prod_{i=1}^T t_i}\right\}(x) = \frac{1}{(2\pi)^{T/2}}\mathrm{p.v.}\int e^{ix\cdot t}\left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \frac{e^{-\sum_{i=1}^T t_i^2/4}}{\prod_{i=1}^T t_i}\,\mathrm{d}t.$$

Let $\gamma(x;\Sigma) \overset{\text{def}}{=} (\det 2\pi\Sigma)^{-T/2}e^{-\frac{1}{2}x^\top\Sigma^{-1}x}$ be the density of $\mathcal{N}(0,\Sigma)$ for *nonsingular* Σ. Note that $\mathcal{F}\{\gamma(x;\Sigma)\}(t) = (2\pi)^{-T/2}e^{-\frac{1}{2}t^\top\Sigma t}$. We thus have

$$\begin{aligned}
&\mathbb{E}\left[\prod_{i=1}^T \phi(x_i) : x\sim\mathcal{N}(0,\Sigma)\right]\\
&= \left\langle \mathcal{F}\left\{\left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\frac{e^{-\sum_{i=1}^T t_i^2/4}}{\prod_{i=1}^T t_i}\right\},\ \gamma(x;\Sigma)\right\rangle\\
&= \left\langle \left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\frac{e^{-\sum_{i=1}^T t_i^2/4}}{\prod_{i=1}^T t_i},\ \mathcal{F}\{\gamma(x;\Sigma)\}\right\rangle\\
&= \left\langle \left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\frac{e^{-\sum_{i=1}^T t_i^2/4}}{\prod_{i=1}^T t_i},\ (2\pi)^{-T/2}e^{-\frac{1}{2}t^\top\Sigma t}\right\rangle\\
&= \left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\int \frac{e^{-\sum_{i=1}^T t_i^2/4}}{\prod_{i=1}^T t_i}(2\pi)^{-T/2}e^{-\frac{1}{2}t^\top\Sigma t}\,\mathrm{d}t\\
&= \left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\int \frac{1}{\prod_{i=1}^T t_i}(2\pi)^{-T/2}e^{-\frac{1}{2}t^\top(\Sigma+\frac{1}{2}I)t}\,\mathrm{d}t\\
&= \left\langle \left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\frac{1}{\prod_{i=1}^T t_i},\ (2\pi)^{-T/2}e^{-\frac{1}{2}t^\top(\Sigma+\frac{1}{2}I)t}\right\rangle\\
&= \left\langle \left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\frac{1}{\prod_{i=1}^T t_i},\ \mathcal{F}\{\gamma(x;\Sigma+\frac{1}{2}I)\}\right\rangle\\
&= \left\langle \mathcal{F}\left\{\left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \mathrm{p.v.}\frac{1}{\prod_{i=1}^T t_i}\right\},\ \gamma(x;\Sigma+\frac{1}{2}I)\right\rangle\\
&= \left\langle \left(\frac{-i\sqrt{2}}{\sqrt{\pi}}\right)^T \left(i\sqrt{\pi/2}\right)^T \prod_{i=1}^T \operatorname{sgn}(x_i),\ \gamma(x;\Sigma+\frac{1}{2}I)\right\rangle
\end{aligned}$$

$$= \mathbb{E}\left[\prod_{i=1}^{T} \operatorname{sgn}(x_i) : x \sim \mathcal{N}(0, \Sigma + \frac{1}{2}I)\right]$$

where we used $\mathcal{F}\{\text{p.v.}t^{-1}\}(x) = i\sqrt{\pi/2}\operatorname{sgn}(x)$. Similar reasoning show that this formula also works when the mean is nonzero:

$$\mathbb{E}\left[\prod_{i=1}^{T} \phi(x_i) : x \sim \mathcal{N}(\mu, \Sigma)\right] = \mathbb{E}\left[\prod_{i=1}^{T} \operatorname{sgn}(x_i) : x \sim \mathcal{N}(\mu, \Sigma + \frac{1}{2}I)\right].$$

A standard continuity argument yields the same formula for singular Σ. Some simple arithmetic reduces the σ case to ϕ.

$$2^{-T}\,\mathbb{E}\left[\prod_{i=1}^{T}(1 + \phi(x_i)) : x \sim \mathcal{N}(\mu, \Sigma)\right] = \mathbb{E}\left[\prod_{i=1}^{T} \mathbb{I}(x_i \geq 0) : x \sim \mathcal{N}(\mu, \Sigma + \frac{1}{2}I)\right].$$

□

C NETSOR$^+$ Master Theorem

In this section, we state the Master Theorem for NETSOR$^+$. Its proof can be found in Appendix I.

We first need to extend the notion of *controlled functions* (Definition 5.3) to functions with parameters, and additionally require a smoothness assumption.

Definition C.1. We say a parametrized function $\phi(-; -) : \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}$ is *parameter-controlled* at $\mathring{\Theta} \in \mathbb{R}^l$ if

1. $\phi(-; \mathring{\Theta})$ is controlled, and
2. there are some controlled $\bar{\phi} : \mathbb{R}^k \to \mathbb{R}$ and some function $f : \mathbb{R}^l \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ that has $f(\mathring{\Theta}) = 0$ and that is continuous at $\mathring{\Theta}$, such that, for all $x^1, \ldots, x^k \in \mathbb{R}$ and $\Theta \in \mathbb{R}^l$,
$$|\phi(x^1, \ldots, x^k; \Theta) - \phi(x^1, \ldots, x^k; \mathring{\Theta})| \leq f(\Theta)\bar{\phi}(x^1, \ldots, x^k).$$

Note that f and $\bar{\phi}$ here can depend on $\mathring{\Theta}$.

Example C.2. Any function that is (pseudo-)Lipschitz[11] in $x^1, \ldots, x^k$ and Θ is parameter-controlled. An example of a discontinuous function that is parameter-controlled is $\phi(x; \theta) = \operatorname{step}(\theta x)$. Then for $\mathring{\theta} \neq 0$,

$$|\phi(x; \theta) - \phi(x; \mathring{\theta})| \leq \frac{|\mathring{\theta} - \theta|}{|\mathring{\theta}|},$$

so we can set $f(\theta) = \frac{|\mathring{\theta} - \theta|}{|\mathring{\theta}|}$ and $\bar{\phi} = 1$ in Definition C.1.

Assumption C.3 (Rank Stability)**.** *For any $W : \mathsf{A}(n, m)$ and any collection $\mathcal{S} \subseteq \{(h : \mathsf{H}(m)) \mid \exists(g : \mathsf{G}(n)), g := Wh\}$, let $H \in \mathbb{R}^{m \times |\mathcal{S}|}$ be the matrix whose columns are $h \in \mathcal{S}$. If $\frac{1}{m}H^\top H \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ converges almost surely to some $\mathring{C}$ as $n, m \to \infty$ with convergent ratio $n/m \to \alpha$, then almost surely* $\operatorname{rank} H = \operatorname{rank} \mathring{C}$ *for all large n and m.*

Note that a common situation where rank stability holds is when all limit $\mathring{C}$ matrices are full rank. By the lower semi-continuity of rank, $\operatorname{rank} H = \operatorname{rank} \mathring{C}$ must hold asymptotically.

[11] A pseudo-Lipschitz function $\phi : \mathbb{R}^r \to \mathbb{R}$ is one that satisfies

$$|\phi(x) - \phi(y)| \leq C\|x - y\|(\|x\|^p + \|y\|^q + 1)$$

for some constants $C, p, q \geq 0$. Roughly speaking, pseudo-Lipschitz functions are those that have polynomially bounded weak derivatives.

Theorem C.4. *Fix any* NETSOR$^+$ *program satisfying Assumption 5.1 and Assumption C.3. Suppose for each parametrized nonlinearity $\phi(-;\Theta)$ in the program (appearing as part of* `Nonlin`$^+$*), the parameters Θ are instantiated with random variables that converge almost surely to some deterministic vector $\mathring{\Theta}$ as $n \to \infty$, and assume ϕ is parameter-controlled at $\mathring{\Theta}$. If $g^1, \ldots, g^M$ are all of the G-vars (including all input G-vars), then for any l, for any random vector $\Theta \in \mathbb{R}^l$ that converges almost surely to a deterministic vector $\mathring{\Theta}$, as $n \to \infty$, and for any $\psi : \mathbb{R}^M \times \mathbb{R}^l \to \mathbb{R}$ parameter-controlled at $\mathring{\Theta}$,*

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g_\alpha^1, \ldots, g_\alpha^M; \Theta) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z; \mathring{\Theta}), \tag{34}$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence, $Z \in \mathbb{R}^M$, and $\mu \in \mathbb{R}^M$ and $\Sigma \in \mathbb{R}^{M\times M}$ are given in Eq. (2), calculated by replacing each parametrized $\phi(-;\Theta)$ with parameterless nonlinearity $\phi(-;\mathring{\Theta})$.

The proof of this theorem can be found in Appendix I.

We will be instantiating the coordinates of Θ typically with "empirical moments"

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g_\alpha^1, \ldots, g_\alpha^M) \tag{35}$$

for some controlled ψ, since such "moments" should converge to a deterministic value by Theorem C.4; or even, recursively,

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g_\alpha^1, \ldots, g_\alpha^M; \Theta') \tag{36}$$

for some sequence of random vectors Θ' that converge a.s. to $\mathring{\Theta}'$ and for ψ parameter-controlled at $\mathring{\Theta}'$. One can keep recursing by replacing Θ' with further empirical moments.

However, there is a slight complication: we are using Theorem C.4 both for the convergence of the parameters in `Nonlin`$^+$ rules in the program, as well as for the convergence Eq. (34), in what seems like could be circular logic. It turns out not hard to straighten out this reasoning, but it requires a bit more notation and setup to state the result. We do this in the next section Appendix C.1, with main theorem Theorem C.11 that will be our primary tool concerning NETSOR$^+$ programs in practice.

To finish up this section, we make several remarks on the assumptions made in Theorem C.4.

Remark C.5 (Necessity of parameter-control). Suppose $\psi(x;\theta) = \mathbb{I}(\theta x \neq 0)$. For $\theta \neq 0$, ψ is 1 everywhere except $\psi(0;\theta) = 0$. For $\theta = 0$, ψ is identically 0. Thus it's easily seen that ψ is not parameter-controlled at $\theta = 0$.

Now, if $g : \mathsf{G}(n)$ is sampled like $g_\alpha \sim \mathcal{N}(0,1)$, then

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g_\alpha;\theta) \xrightarrow{\text{a.s.}} 1$$

if $\theta = 1/n$ so that $\theta \to \mathring{\theta} = 0$, but

$$\mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z;\mathring{\theta}) = \mathbb{E}\, 0 = 0.$$

So our Master Theorem can't hold in this case.

Remark C.6 (Necessity of Rank Stability Assumption Assumption C.3). Suppose we have two input G-vars $g^1, g^2 : \mathsf{G}(n)$ which are sampled independently as $g_\alpha^1, g_\alpha^2 \sim \mathcal{N}(0,1)$. Let $W : \mathsf{A}(n,n)$ be sampled as $W_{\alpha\beta} \sim \mathcal{N}(0, 1/n)$. Then we can define $h^2 := \theta g^2 : \mathsf{H}(n)$ where $\theta = \exp(-n)$ as a function of n, using `Nonlin`$^+$, so that $h_\alpha^2 \xrightarrow{\text{a.s.}} 0$. Additionally, let $\bar{g}^1 := Wg^1 : \mathsf{G}(n)$ and $\bar{g}^2 := Wh^2 : \mathsf{G}(n)$. Again, $\bar{g}_\alpha^2 \xrightarrow{\text{a.s.}} 0$ but for any finite n, $\bar{g}^2$ is linearly independent from $\bar{g}^1$. Thus rank stability does not hold here.

Now consider the (parameterless) nonlinearity $\psi(x,y)$ that is 1 except on the line $y = 0$, where it is 0. Then

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(\bar{g}_\alpha^1, \bar{g}_\alpha^2) \xrightarrow{\text{a.s.}} 1$$

but
$$\mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z^{\bar{g}^1}, Z^{\bar{g}^2}) = \mathbb{E}\, 0 = 0.$$

Remark C.7 (Rank Stability Already Holds for NETSOR Programs). It turns out that, as long as we only have parameterless nonlinearities, we get rank stability Assumption C.3 for free. This is formulated explicitly in Lemma H.4. It is as a result of our proof of Theorem 5.4 that interleaves an inductive proof of this rank stability (more generally, the inductive hypothesis CoreSet) with an inductive proof of the "empirical moment" convergence (the inductive hypothesis Moments).

C.1 Self-Parametrized NETSOR$^+$ Programs and Their Master Theorem

As stated below Theorem C.4, there could be potentially circular logic when allowing `Nonlin`$^+$ rules to take parameters depending on previously defined variables in the program, such as in the form of Eq. (35). In this section, we untangle this potentially circular logic into a sound reasoning

- by introducing a scalar type into NETSOR$^+$ programs to explicitly extract the `Nonlin`$^+$ parameters into their own variables (Definition C.8). These scalar variables can recursively depend on previously defined scalar variables, making the "recursive parameters" discussed in Eq. (36) much more succinctly and clearly expressed.
- and by proving a Master Theorem for such NETSOR$^+$ programs (Theorem C.11). This theorem will be the primary way through which we analyze NETSOR$^+$ programs in practice.

Definition C.8. [12] A *self-parametrized* NETSOR$^+$ *program* is a NETSOR program where we have an additional scalar type, called C, which should intuitively be thought of as random variables that tend to a deterministic limit (i.e. a C*onstant*) almost surely. Colloquially, we will call variables of type C "C-vars." C-vars can be used as parameters of nonlinearities in `Nonlin`$^+$ rules, hence the *"self-parametrized"* in the name.

For completeness, we specify a self-parametrized NETSOR$^+$ program as follows:

Input A set of input C-vars, in addition to the G- and A-vars allowed in Definition 4.1.

Body New variables can be introduced and assigned via the following rules

`MatMul` Same as in Definition 4.1.

`LinComb` Same as in Definition 4.1.

`Nonlin`$^+$ If $x^1, \ldots, x^k : \mathsf{G}(n)$ are G-vars with the same dimension n, $\theta_1, \ldots, \theta_l : \mathsf{C}$ are C-vars, and $\phi(-;-) : \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}$ is a parametrized function, then we may create an H-var
$$\phi(x^1, \ldots, x^k; \theta_1, \ldots, \theta_l) : \mathsf{H}(n)$$
where $\phi(-;\theta_1, \ldots, \theta_l)$ acts coordinatewise.

`Moment` If $x^1, \ldots, x^k : \mathsf{G}(n)$ are G-vars with the same dimension n, $\theta_1, \ldots, \theta_l : \mathsf{C}$ are C-vars, and $\phi(-;-) : \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}$ is a parametrized function, then we may create a C-var
$$\frac{1}{n}\sum_{\alpha=1}^{n} \phi(x^1_\alpha, \ldots, x^k_\alpha; \theta_1, \ldots, \theta_l) : \mathsf{C}.$$

Output Same as in Definition 4.1.

The self-parametrized NETSOR$^+$ programs we are concerned will have all of its C-vars convergent to a deterministic constant. We thus need this to be true for the input C-vars at the very least. We encapsulate this requirement below.

Assumption C.9. *Fix a self-parametrized* NETSOR$^+$ *program satisfying Assumption 5.1. Assume each input C-var θ is sampled in a way such that $\theta \xrightarrow{\text{a.s.}} \mathring{\theta}$ as $n \to \infty$ for some deterministic scalar $\mathring{\theta} \in \mathbb{R}$.*

[12] We keep the definition here informal in terms of programming language convention to be accessible to the general machine learning audience. For those with PL background, see Appendix J.

Now, we shall define μ and Σ for self-parametrized NETSOR$^+$ programs just as in NETSOR$^+$ programs. The only complication here is that we also need to keep track of the limit values of the C-vars in order to do so. See Definition C.10 below.

Definition C.10. Fix a NETSOR$^+$ program with scalar variables satisfying Assumption C.9. For the purpose of this definition, write $g^1, \dots, g^M$ for the entirety of the G-vars in the program, including input G-vars.

New Notations For each H-var h introduced by `Nonlin`$^+$, *we introduce the notations* $\varphi^h, \Theta^h, \vartheta^h_i, \ell^h$ *as follows*: denote the associated parametrized nonlinearity by $\varphi^h(-;-) : \mathbb{R}^M \times \mathbb{R}^{\ell^h} \to \mathbb{R}$ (implicitly padded so that it has as many input slots as G-vars in the program) and its parameters by $\Theta^h = (\vartheta^h_1, \dots, \vartheta^h_{\ell^h}) \in \mathbb{R}^{\ell^h}$ with length ℓ^h. For each G-var g^i, we also set $\varphi^{g^i}(x^1, \dots, x^M) = x^i$ and $\Theta^{g^i} = () \in \mathbb{R}^0$ to be the empty vector (so that $\ell^{g^i} = 0$).

Likewise, for each C-var θ introduced by `Moment`, *we introduce the notations* $\varphi^\theta, \Theta^\theta, \vartheta^\theta_i, \ell^\theta$ *as follows*: denote the associated parametrized nonlinearity by $\varphi^\theta(-;-) : \mathbb{R}^M \times \mathbb{R}^{\ell^\theta} \to \mathbb{R}$ (implicitly padded so that it has as many input slots as G-vars in the program) and its parameters by $\Theta^\theta = (\vartheta^\theta_1, \dots, \vartheta^\theta_{\ell^\theta}) \in \mathbb{R}^{\ell^\theta}$ with length ℓ^θ.

Extending the $(\mathring{\ })$ *notation from Assumption C.9 and the Recursive Definition of* μ *and* Σ Given μ^{in} and Σ^{in} as in Assumption 5.1, we define μ and Σ on G-vars, along with "limit scalars" $\mathring{\theta}$ for each C-var θ (extending $\mathring{\theta}$ given by Assumption C.9 for input θ), as follows: For any pair of G-vars g, g' (among $g^1, \dots, g^M$), we define recursively

$$\mu(g) \overset{\text{def}}{=} \begin{cases} \mu^{\text{in}}(g) & \text{if } g \text{ is input} \\ \sum_i a_i \mu(y^i) & \text{if } g = \sum_i a_i y^i, \text{ (}\texttt{LinComb}\text{)} \\ 0 & \text{otherwise} \end{cases}$$

$$\Sigma(g, g') \overset{\text{def}}{=} \begin{cases} \Sigma^{\text{in}}(g, g') & \text{if } g, g' \text{ are inputs} \\ \sum_i a_i \Sigma(y^i, g') & \text{if } g = \sum_i a_i y^i, \text{ (}\texttt{LinComb}\text{)} \\ \sum_i a_i \Sigma(g, y^i) & \text{if } g' = \sum_i a_i y^i, \text{ (}\texttt{LinComb}\text{)} \\ \sigma_W^2 \mathbb{E}_Z \varphi^h(Z; \mathring{\Theta}^h) \varphi^{h'}(Z; \mathring{\Theta}^{h'}) & \text{if } g = Wh, g' = Wh', \text{ (}\texttt{MatMul}\text{)} \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

(this is the same as Eq. (2) except the `MatMul` case) and for each C-var θ introduced by `Moment`,

$$\mathring{\theta} \overset{\text{def}}{=} \mathop{\mathbb{E}}_{Z} \varphi^\theta(Z; \mathring{\Theta}^\theta). \tag{38}$$

In all of the equations above, $Z \sim \mathcal{N}(\mu, \Sigma)$ is a random Gaussian vector with an entry for each G-var in the program, and $\mathring{\Theta}^u$ denotes $(\mathring{\vartheta}^u_1, \dots, \mathring{\vartheta}^u_{\ell^u})$, for H-var or C-var u.

Note that since φ^h, $\varphi^{h'}$, and φ^θ only depend on entries of Z corresponding to G-vars previous to h, h', or θ, the expectations involving Z only depend on entries of μ and Σ already defined, so there is no circular logic in this recursive definition of μ and Σ.

Note that the notation φ^h will be overloaded in a semantically consistent way in the context of NETSOR$\circ$ programs; see Definition E.6.

We are finally ready to formulate the Master Theorem for self-parametrized NETSOR$^+$ programs, which basically is just Theorem C.4 but explicitly allowing parameters of the form Eq. (35) ("empirical moments") in `Nonlin`$^+$.

Theorem C.11 (Self-Parameterized NETSOR$^+$ Master Theorem)**.** *Fix any self-parametrized* NETSOR$^+$ *program satisfying Assumption C.9 and Assumption C.3. For H-var or C-var u, adopt the notation φ^u, Θ^u, ℓ^u from Definition C.10 and also let $\mu, \Sigma, \mathring{\theta}$ be as computed in Definition C.10. Let $g^1, \dots, g^M$ be all of the G-vars in the program (including all input G-vars).*

Suppose for every H-var or C-var u, $\varphi^u(-;\Theta^u)$ is parameter-controlled at $\mathring{\Theta}^u$.

1. *Then for any l, for any random vector $\Theta \in \mathbb{R}^l$ that converges almost surely to a deterministic vector $\mathring{\Theta}$, as $n \to \infty$, and for any $\psi(-;-) : \mathbb{R}^M \times \mathbb{R}^l \to \mathbb{R}$ parameter-controlled at $\mathring{\Theta}$,*

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g^1_\alpha, \ldots, g^M_\alpha; \Theta) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z; \mathring{\Theta}),$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence.

2. *In addition, for each C-var θ in the program,*

$$\theta \xrightarrow{\text{a.s.}} \mathring{\theta}.$$

This theorem almost trivially follows from Theorem C.4, since the parameter vectors Θ^h intuitively should converge to deterministic limits $\mathring{\Theta}^h$. The only slight complication is that this convergence intuitvely follows from Theorem C.4 itself in what may be a circular logic, so we need to be slightly careful to unwind this logic into a valid inductive argument. We do so below, assuming Theorem C.4 (which is proved in Appendix I).

Proof. Notice that the 2nd claim about $\theta \xrightarrow{\text{a.s.}} \mathring{\theta}$ follows immediately from the 1st claim, so we will prove the 1st claim here.

Assume that the G-vars $g^1, \ldots, g^M$ are in order of appearance in the program, and that $g^1, \ldots, g^{m_0}$ (with $m_0 \le M$) are all of the input G-vars. We perform simultaneous induction on two claims Moments(m) and CVarLimits(m) in m, defined below

Moments (m) *For any l, for any random vector $\Theta \in \mathbb{R}^l$ that converges almost surely to a deterministic vector $\mathring{\Theta}$ as $n \to \infty$, and for any $\psi(-;-) : \mathbb{R}^m \times \mathbb{R}^l \to \mathbb{R}$ parameter-controlled at $\mathring{\Theta}$,*

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g^1_\alpha, \ldots, g^m_\alpha; \Theta) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu|_m,\Sigma|_m)} \psi(Z; \mathring{\Theta})$$

where $\mu|_m$ and $\Sigma|_m$ are the restriction of μ and Σ to $g^1, \ldots, g^m$.

CVarLimits (m) *For each C-var θ introduced before g^m,*

$$\theta \xrightarrow{\text{a.s.}} \mathring{\theta}$$

as $n \to \infty$, where $\mathring{\theta}$ is as computed in Definition C.10.

When $m = M$, we would have Theorem C.11 by Moments(M).

Base case: $m = m_0$ (input G-vars only). Moments(m_0) trivially follows from Theorem C.4. CVarLimits(m_0) follows from Assumption C.9.

Now suppose Moments(m) and CVarLimits(m) are true; we aim to show Moments$(m+1)$ and CVarLimits$(m+1)$.

Inductive case: CVarLimits$(m+1)$ By CVarLimits(m), it suffices to show $\theta \xrightarrow{\text{a.s.}} \mathring{\theta}$ for all θ introduced after g^m but before g^{m+1}. We do so by another induction (an *inner induction*) in order of C-var appearance.

The **inner base case** is the first C-var θ introduced after g^m. Its parameters Θ^θ are among those introduced before g^m, so by induction hypothesis CVarLimits(m),

$$\Theta^\theta \xrightarrow{\text{a.s.}} \mathring{\Theta}^\theta.$$

By the assumption of Theorem C.11 that φ^θ is parameter-controlled at $\mathring{\Theta}^\theta$, we have

$$\theta = \frac{1}{n}\sum_{\alpha=1}^{n} \varphi^\theta(g^1_\alpha, \ldots, g^m_\alpha; \Theta^\theta) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu|_m,\Sigma|_m)} \varphi^\theta(Z; \mathring{\Theta}^\theta) = \mathring{\theta}$$

by induction hypothesis Moments(m) (where we have explicitly truncated the input slots of φ^θ to reflect its dependence only on $g^1, \ldots, g^m$). The **inner inductive case**, for a later θ, follows the same logic, once we assume the inner inductive hypothesis that each θ' introduced before θ has $\theta' \xrightarrow{\text{a.s.}} \mathring{\theta}'$.

Inductive case: Moments$(m+1)$ The claim is trivially true by Moments(m) if g^{m+1} is introduced via `LinComb`, so consider the case when g^{m+1} is introduced via `MatMul`

$$g^{m+1} := Wh$$

where $h : \mathsf{H}(n)$ is an H-var with associated nonlinearity φ^h and parameters Θ^h as defined in Definition C.10. By the claim CVarLimits$(m+1)$ we proved above, $\Theta^h \xrightarrow{\text{a.s.}} \mathring{\Theta}^h$. By the assumption of Theorem C.11, φ^h is parameter-controlled at the parameter limit $\mathring{\Theta}^h$. Thus, the subprogram up to and including the introduction of g^{m+1} satisfies the assumptions of Theorem C.4. Consequently, Moments$(m+1)$ is true by Theorem C.4.

This completes the simultaneous induction of Moments and CVarLimits and thus the proof of Theorem C.11. □

C.2 Gaussian Process Behavior of NETSOR$^+$ Programs

We can generalize the Gaussian process behavior (Corollary 5.5) to cases involving `Nonlin`$^+$:

Corollary C.12 (Computing the GP Kernel for NETSOR$^+$ programs)**.** *Adopt the same assumptions and notations as in Theorem C.4. Suppose the program outputs $(v^\top x^1/\sqrt{n}, \ldots, v^\top x^k/\sqrt{n})$, where*

- *$v : \mathsf{G}(n), v_\alpha \sim \mathcal{N}(0, \sigma_v^2)$, is an input G-var not used elsewhere in the program and is sampled independently from all other G-vars, and*
- *x^i was introduced as $x^i := \phi^i(g^1, \ldots, g^M; \Theta^i)$ for parametrized nonlinearity ϕ^i and parameter vector Θ^i that converges a.s. to a deterministic vector $\mathring{\Theta}^i$ as $n \to \infty$. Assume ϕ^i is parameter-controlled at $\mathring{\Theta}^i$.*

Then the output vector converges in distribution to $\mathcal{N}(0, K)$ where

$$K_{ij} = \sigma_v^2 \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \phi^i(Z; \mathring{\Theta}^i)\phi^j(Z; \mathring{\Theta}^j) \tag{39}$$

with μ, Σ computed by replacing each parametrized $\phi(-; \Theta)$ with the parameterless $\phi(-; \mathring{\Theta})$ in Eq. (2), as in Theorem C.4.

The proof is a straightforward application of Theorem E.8 and Proposition G.4. Likewise, for self-parametrized programs, we have a similar result:

Corollary C.13 (Computing the GP Kernel for self-parametrized NETSOR$^+$ programs)**.** *Adopt the same assumptions and notations as in Theorem C.11. Suppose the program outputs $(v^\top x^1/\sqrt{n}, \ldots, v^\top x^k/\sqrt{n})$, where*

- *$v : \mathsf{G}(n), v_\alpha \sim \mathcal{N}(0, \sigma_v^2)$, is an input G-var not used elsewhere in the program and is sampled independently from all other G-vars, and*
- *x^i was introduced as $x^i := \varphi^{x^i}(g^1, \ldots, g^M; \Theta^{x^i})$ for self-parametrized nonlinearity φ^{x^i} and parameter vector Θ^{x^i} (composed of C-vars) as defined in Definition C.10. Let $\mathring{\Theta}^{x^i}$ be the limit parameter as in Definition C.10. Note that φ^{x^i} is parameter-controlled at $\mathring{\Theta}^{x^i}$ by assumption of Theorem C.11.*

Then the output vector converges in distribution to $\mathcal{N}(0, K)$ where

$$K_{ij} = \sigma_v^2 \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \varphi^{x^i}(Z; \mathring{\Theta}^{x^i})\varphi^{x^j}(Z; \mathring{\Theta}^{x^j}), \quad \textit{with } \mu, \Sigma \textit{ defined in Eq. (37).} \tag{40}$$

D Example GP Kernel Computation with NETSOR$^+$

D.1 Layernorm: Concrete Example (Program 9)

Consider the example layernorm network in Program 9. This is a self-parametrized NETSOR$^+$ program.

Self-parametrized NETSOR$^+$ program 9 Layernorm Network

```
Input: $W^1x, W^1x' : \mathsf{G}(n)$
Input: $W^2 : \mathsf{G}(n)$
Input: $v : \mathsf{G}(n)$
  // Mean and variance of $W^1x$
  // Moment
  $\nu^1 := \frac{1}{n}\sum_{\alpha=1}^n (W^1x)_\alpha : \mathsf{C}$
  $var^1 := \frac{1}{n}\sum_{\alpha=1}^n (W^1x)_\alpha^2 - (\nu^1)^2 : \mathsf{C}$
  // Nonlin$^+$
  $x^1 := \mathrm{ReLU}\left(\frac{(W^1x)-\nu^1\mathbf{1}}{\sqrt{var^1}}\right) : \mathsf{H}(n)$
  $h^2 := W^2x^1 : \mathsf{G}(n)$
  // Mean and variance of $h^2$
  // Moment
  $\nu^2 := \frac{1}{n}\sum_{\alpha=1}^n h_\alpha^2 : \mathsf{C}$
  $var^2 := \frac{1}{n}\sum_{\alpha=1}^n (h_\alpha^2)^2 - (\nu^2)^2 : \mathsf{C}$
  // Nonlin$^+$
  $x^2 := \mathrm{ReLU}\left(\frac{h^2-\nu^2\mathbf{1}}{\sqrt{var^2}}\right) : \mathsf{H}(n)$
  // Same thing for $x'$
  // Mean and variance of $W^1x'$
  // Moment
  $\nu^{1\prime} := \frac{1}{n}\sum_{\alpha=1}^n (W^1x')_\alpha : \mathsf{C}$
  $var^{1\prime} := \frac{1}{n}\sum_{\alpha=1}^n (W^1x')_\alpha^2 - (\nu^{1\prime})^2 : \mathsf{C}$
  // Nonlin$^+$
  $x^{1\prime} := \mathrm{ReLU}\left(\frac{(W^1x')-\nu^{1\prime}\mathbf{1}}{\sqrt{var^{1\prime}}}\right) : \mathsf{H}(n)$
  $h^{2\prime} := W^2x^{1\prime} : \mathsf{G}(n)$
  // Mean and variance of $h^{2\prime}$
  // Moment
  $\nu^{2\prime} := \frac{1}{n}\sum_{\alpha=1}^n h_\alpha^{2\prime} : \mathsf{C}$
  $var^{2\prime} := \frac{1}{n}\sum_{\alpha=1}^n (h_\alpha^{2\prime})^2 - (\nu^{2\prime})^2 : \mathsf{C}$
  // Nonlin$^+$
  $x^{2\prime} := \mathrm{ReLU}\left(\frac{h^{2\prime}-\nu^{2\prime}\mathbf{1}}{\sqrt{var^{2\prime}}}\right) : \mathsf{H}(n)$
Output: $(v^\top x^2/\sqrt{n}, v^\top x^{2\prime}/\sqrt{n})$
```

Setup Suppose the inputs $x, x' \neq 0$ are in $\mathbb{R}^m$. The network has parameters $W^1 \in \mathbb{R}^{n\times m}, W^2 \in \mathbb{R}^{n\times n}$, and $v \in \mathbb{R}^n$. Let us sample them as follows

$$W_{\alpha\beta}^1 \sim \mathcal{N}(0, \sigma_w^2/m), \quad W_{\alpha\beta}^2 \sim \mathcal{N}(0, \sigma_w^2/n), \quad v_\alpha \sim \mathcal{N}(0, \sigma_v^2),$$

for $\sigma_w, \sigma_v > 0$. This corresponds to the NETSOR$^+$ sampling data $\mu^{\text{in}} = 0$ and Σ^{in} given as

$$\Sigma^{\text{in}}(W^1x, W^1x) = \sigma_w^2\|x\|^2/m, \quad \Sigma^{\text{in}}(W^1x, W^1x') = \sigma_w^2 x^\top x'/m, \quad \Sigma^{\text{in}}(W^1x', W^1x') = \sigma_w^2\|x'\|^2/m,$$

$\Sigma^{\text{in}}(v, v) = \sigma_v^2$, and $\Sigma^{\text{in}}(g, g') = 0$ for any other pairs of input G-vars g, g'.

D.1.1 Computing μ, Σ, and Limit Parameters $\mathring{\theta}$

Let's compute the values of μ, Σ, $\mathring{\theta}$ in order of the appearance of the variables, according to Definition C.10. For each C-var or H-var u, we also show that φ^u is parameter-controlled at $\mathring{\vartheta}^u$.

First, one can quickly notice that $\mu(g) = 0$ for all G-vars g, so we shall focus on computing Σ and $\mathring{\theta}$.

C-var ν^1 Here we have introduced ν^1 via Moment by

$$\nu^1 := \frac{1}{n}\sum_{\alpha=1}^n \varphi^{\nu^1}((W^1x)_\alpha), \quad \text{where } \varphi^{\nu^1}(z) = z,$$

and there are no parameters. The function φ^{ν^1} is then obviously controlled and trivially parameter-controlled. Finally, by Eq. (38), we set

$$\mathring{\nu}^1 \overset{\text{def}}{=} \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,\sigma_w^2\|x\|^2/m)} z = 0.$$

C-var var^1 Note here

$$var^1 := \frac{1}{n}\sum_{\alpha=1}^n \varphi^{var^1}((W^1x)_\alpha; \nu^1), \quad \text{where } \varphi^{var^1}(z;\theta) \overset{\text{def}}{=} z^2 - \theta^2.$$

(Here superscript-2 denotes square, not an index). Since $\varphi^{var^1}(-;-)$ is pseudo-Lipschitz in both its inputs and its parameter jointly, it is parameter-controlled at $\theta = \mathring{\nu}^1 = 0$ by Example C.2.

In addition, $\mathring{var}^1$ is computed by Eq. (38) as

$$\mathring{var}^1 \overset{\text{def}}{=} \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,\sigma_w^2\|x\|^2/m)} \varphi^{var^1}(z;\mathring{\nu}^1) = \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,\sigma_w^2\|x\|^2/m)} z^2 = \frac{\sigma_w^2}{m}\|x\|^2.$$

H-var x^1 The first H-var introduced in the program is $x^1 := \mathrm{ReLU}\left(\frac{(W^1x)-\nu^1\mathbf{1}}{\sqrt{var^1}}\right)$. It can be written as a Nonlin$^+$ with

$$x^1 := \varphi^{x^1}(W^1x; \nu^1, var^1)$$

where $\varphi^{x^1}(-;-) : \mathbb{R} \times \mathbb{R}^2 \to \mathbb{R}$, and

$$\varphi^{x^1}(z; \theta_1, \theta_2) \stackrel{\text{def}}{=} \mathrm{ReLU}\left(\frac{z-\theta_1}{\sqrt{\theta_2}}\right).$$

Since $\sigma_w > 0$ and $x \neq 0$, we have $\mathring{var}^1 \neq 0$, and we claim that φ^{x^1} is parameter-controlled at $(\mathring{\nu}^1, \mathring{var}^1) = \left(0, \frac{\sigma_w^2}{m}\|x\|^2\right)$.

Indeed, $\varphi^{x^1}(-; \mathring{\nu}^1, \mathring{var}^1)$ is obviously controlled, so that condition 1 of Definition C.1 is satisfied. In addition, for any $z \in \mathbb{R}$,

$$\begin{aligned}
\left|\varphi^{x^1}(z;\theta_1,\theta_2) - \varphi^{x^1}(z;\mathring{\nu}^1,\mathring{var}^1)\right| &= \left|\mathrm{ReLU}\left(\frac{z-\theta_1}{\sqrt{\theta_2}}\right) - \mathrm{ReLU}\left(\frac{z-\mathring{\nu}^1}{\sqrt{\mathring{var}^1}}\right)\right| \\
&\leq \left|\frac{z-\theta_1}{\sqrt{\theta_2}} - \frac{z-\mathring{\nu}^1}{\sqrt{\mathring{var}^1}}\right| \\
&= \left|z\left(\frac{1}{\sqrt{\theta_2}} - \frac{1}{\sqrt{\mathring{var}^1}}\right) - \left(\frac{\theta_1}{\sqrt{\theta_2}} - \frac{\mathring{\nu}^1}{\sqrt{\mathring{var}^1}}\right)\right| \\
&\leq \left|z\left(\frac{1}{\sqrt{\theta_2}} - \frac{1}{\sqrt{\mathring{var}^1}}\right)\right| + \left|\frac{\theta_1}{\sqrt{\theta_2}} - \frac{\mathring{\nu}^1}{\sqrt{\mathring{var}^1}}\right| \\
&\leq \sqrt{\left(\frac{1}{\sqrt{\theta_2}} - \frac{1}{\sqrt{\mathring{var}^1}}\right)^2 + \left(\frac{\theta_1}{\sqrt{\theta_2}} - \frac{\mathring{\nu}^1}{\sqrt{\mathring{var}^1}}\right)^2}\sqrt{z^2+1}
\end{aligned}$$

by Cauchy-Schwarz. Note that $\sqrt{\left(\frac{1}{\sqrt{\theta_2}} - \frac{1}{\sqrt{\mathring{var}^1}}\right)^2 + \left(\frac{\theta_1}{\sqrt{\theta_2}} - \frac{\mathring{\nu}^1}{\sqrt{\mathring{var}^1}}\right)^2}$ equals 0 and is continuous at $(\theta_1, \theta_2) = (\mathring{\nu}^1, \mathring{var}^1)$ because $\mathring{var}^1 \neq 0$. Then since $\sqrt{z^2+1}$ is controlled in z, φ^{x^1} satisfies property 2 of Definition C.1. Altogther, we have shown that φ^{x^1} is indeed parameter-controlled at $(\mathring{\nu}^1, \mathring{var}^1)$.

G-var h^2 By the MatMul case of Eq. (37),

$$\Sigma(h^2,h^2) = \sigma_w^2 \mathop{\mathbb{E}}_z \phi(z;\mathring{\nu}^1,\mathring{var}^1)^2, \qquad \Sigma(h^2, W^1x) = \Sigma(h^2, W^1x') = 0,$$

where $z \sim \mathcal{N}(\mu(W^1x), \Sigma(W^1x, W^1x)) = \mathcal{N}(0, \frac{\sigma_w^2}{m}\|x\|^2)$, and

$$\phi(z;\theta_1,\theta_2) \stackrel{\text{def}}{=} \varphi^{x^1}(z;\theta_1,\theta_2) = \mathrm{ReLU}\left(\frac{z-\theta_1}{\sqrt{\theta_2}}\right).$$

We can then simplify

$$\Sigma(h^2,h^2) = \sigma_w^2 \mathop{\mathbb{E}}_z \mathrm{ReLU}\left(\frac{z}{\sqrt{\frac{\sigma_w^2}{m}\|x\|^2}}\right)^2 = \sigma_w^2 \mathop{\mathbb{E}}_{\zeta\sim\mathcal{N}(0,1)} \mathrm{ReLU}(\zeta)^2 = \frac{1}{2}\sigma_w^2.$$

C-var ν^2 Similar to the case of ν^1, we can express ν^2 via Moment by

$$\nu^2 := \frac{1}{n}\sum_{\alpha=1}^n \varphi^{\nu^2}(h_\alpha^2), \quad \text{where } \varphi^{\nu^2}(z) = z,$$

and there are no parameters. The function φ^{ν^2} is then obviously controlled and trivially parameter-controlled. Finally, by Eq. (38), we set

$$\mathring{\nu}^2 \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{z\sim\mathcal{N}(\mu(h^2),\Sigma(h^2,h^2))} z = \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,\frac{1}{2}\sigma_w^2)} z = 0.$$

C-var var^2 Similar to the case of var^1, we can express var^2 via `Moment` by

$$var^2 := \frac{1}{n}\sum_{\alpha=1}^{n} \varphi^{var^2}(h^2_\alpha; \nu^2), \quad \text{where } \varphi^{var^2}(z;\theta) \stackrel{\text{def}}{=} z^2 - \theta^2.$$

(Here z^2 and θ^2 are the squares of z and θ). Since $\varphi^{var^2}(-;-)$ is pseudo-Lipschitz in both its inputs and its parameter jointly, it is parameter-controlled at $\theta = \mathring{\nu}^2 = 0$ by Example C.2.

In addition, $\mathring{var}^2$ is computed by Eq. (38) as

$$\mathring{var}^2 \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{z\sim\mathcal{N}(\mu(h^2),\Sigma(h^2,h^2))} \varphi^{var^2}(z;\mathring{\nu}^2) = \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,\frac{1}{2}\sigma_w^2)} z^2 = \frac{1}{2}\sigma_w^2.$$

H-var x^2 Similar to the case of x^1, we can express x^2 via `Nonlin`$^+$ by

$$x^2 := \varphi^{x^2}(h^2; \nu^2, var^2)$$

where $\varphi^{x^2}(-;-) : \mathbb{R}\times\mathbb{R}^2 \to \mathbb{R}$, and

$$\varphi^{x^2}(z;\theta_1,\theta_2) \stackrel{\text{def}}{=} \mathrm{ReLU}\left(\frac{z-\theta_1}{\sqrt{\theta_2}}\right).$$

Since $\sigma_w > 0$, we also have $\mathring{var}^2 > 0$. Then by the same reasoning as in the case of x^1, φ^{x^2} is parameter-controlled at $\mathring{\Theta}^{x^2} = (\mathring{\nu}^2, \mathring{var}^2)$.

C-vars $\nu^{1\prime}, var^{1\prime}$ **and H-var** $x^{1\prime}$ These calculations proceed similarly to those for ν^1, var^1 and x^1. We end up with

$$\mathring{\nu}^{1\prime} = 0, \quad \mathring{var}^{1\prime} = \frac{\sigma_w^2}{m}\|x'\|^2,$$

and, for each $u \in \{\nu^{1\prime}, var^{1\prime}, x^{1\prime}\}$, the associated nonlinearity φ^u is parameter-controlled at limit parameter Θ^u.

G-var $h^{2\prime}$ By the `MatMul` case of Eq. (37),

$$\Sigma(h^2,h^{2\prime}) = \sigma_w^2 \mathop{\mathbb{E}}_{z,z'} \phi(z;\mathring{\nu}^1,\mathring{var}^1)\phi(z;\mathring{\nu}^{1\prime},\mathring{var}^{1\prime}), \quad \Sigma(h^{2\prime},h^{2\prime}) = \sigma_w^2 \mathop{\mathbb{E}}_{z'} \phi(z';\mathring{\nu}^{1\prime},\mathring{var}^{1\prime})^2$$

and $\Sigma(h^{2\prime}, g) = 0$ for all other G-var g (by the "otherwise" case of Eq. (37)), where

$$(z,z') \sim \mathcal{N}(\mu|_{W^1x,W^1x'}, \Sigma|_{W^1x,W^1x'}) = \mathcal{N}\left(0, \frac{\sigma_w^2}{m}\begin{pmatrix}\|x\|^2 & x^\top x' \\ x^\top x' & \|x'\|^2\end{pmatrix}\right)$$

and

$$\phi(z;\theta_1,\theta_2) \stackrel{\text{def}}{=} \varphi^{x^1}(z;\theta_1,\theta_2) = \varphi^{x^{1\prime}}(z;\theta_1,\theta_2) = \mathrm{ReLU}\left(\frac{z-\theta_1}{\sqrt{\theta_2}}\right).$$

We can simplify

$$\begin{aligned}\Sigma|_{h^2,h^{2\prime}} &= \sigma_w^2 \mathop{\mathbb{E}}_{\tilde{z},\tilde{z}'} \mathrm{ReLU}(\tilde{z})\mathrm{ReLU}(\tilde{z}'), \quad (\tilde{z},\tilde{z}') \sim \mathcal{N}\left(0, \begin{pmatrix} 1 & \frac{x^\top x'}{\|x\|\|x'\|} \\ \frac{x^\top x'}{\|x\|\|x'\|} & 1\end{pmatrix}\right) \\ &= \sigma_w^2 \mathrm{V}_{\mathrm{ReLU}}\begin{pmatrix} 1 & \frac{x^\top x'}{\|x\|\|x'\|} \\ \frac{x^\top x'}{\|x\|\|x'\|} & 1\end{pmatrix},\end{aligned}$$

where $\mathrm{V}_{\mathrm{ReLU}}$ is as given in Fact B.2. In particular, with $c \stackrel{\text{def}}{=} \frac{x^\top x'}{\|x\|\|x'\|}$, this yields

$$\Sigma(h^{2\prime},h^{2\prime}) = \Sigma(h^2,h^2) = \frac{1}{2}\sigma_w^2, \quad \Sigma(h^2,h^{2\prime}) = \frac{\sigma_w^2}{2\pi}(\sqrt{1-c^2} + (\pi - \arccos c)c). \tag{41}$$

C-vars $\nu^{2\prime}, var^{2\prime}$ and H-var $x^{2\prime}$ These calculations proceed similarly to those for ν^2, var^2 and x^2. We end up with

$$\mathring{\nu}^{2\prime} = 0, \quad \mathring{var}^{2\prime} = \frac{1}{2}\sigma_w^2,$$

and, for each $u \in \{\nu^{2\prime}, var^{2\prime}, x^{2\prime}\}$, the associated nonlinearity φ^u is parameter-controlled at limit parameter Θ^u.

D.1.2 Computing the GP Kernel

It is easy to see that the set of H-vars are all linearly independent almost surely. Therefore we may apply Corollary C.13. By Corollary C.13, $(v^\top x^2/\sqrt{n}, v^\top x^{2\prime}/\sqrt{n})$ converges in distribution to $\mathcal{N}(0, K)$ where

$$K = \sigma_v^2 \mathop{\mathbb{E}}_{z,z'} \begin{pmatrix} \phi(z; \mathring{\Theta}^{x^2})^2 & \phi(z; \mathring{\Theta}^{x^2})\phi(z; \mathring{\Theta}^{x^{2\prime}}) \\ \phi(z; \mathring{\Theta}^{x^2})\phi(z; \mathring{\Theta}^{x^{2\prime}}) & \phi(z'; \mathring{\Theta}^{x^{2\prime}})^2 \end{pmatrix}$$

where $\phi(z; \theta_1, \theta_2) \overset{\text{def}}{=} \text{ReLU}\left(\frac{z-\theta_1}{\sqrt{\theta_2}}\right)$ and $(z, z') \sim \mathcal{N}(\mu|_{h^2,h^{2\prime}}, \Sigma|_{h^2,h^{2\prime}})$ with $\mu|_{h^2,h^{2\prime}} = 0$ and $\Sigma|_{h^2,h^{2\prime}}$ given in Eq. (41). Since $\mathring{\vartheta}_1^{x^2} = \mathring{\nu}^2 = \mathring{\vartheta}_1^{x^{2\prime}} = \mathring{\nu}^{2\prime} = 0$ and $\mathring{\vartheta}_2^{x^2} = \mathring{var}^2 = \mathring{\vartheta}_2^{x^{2\prime}} = \mathring{var}^{2\prime} = \frac{1}{2}\sigma_w^2$, we can simplify

$$K = \sigma_v^2 \mathrm{V}_{\text{ReLU}}\left((\sigma_w^2/2)^{-1}\Sigma|_{h^2,h^{2\prime}}\right) = \frac{2\sigma_v^2}{\sigma_w^2}\mathrm{V}_{\text{ReLU}}\left(\Sigma|_{h^2,h^{2\prime}}\right).$$

D.2 Layernorm: General Case

As mentioned in Appendix A, layernorm in general can be implemented with `Nonlin`$^+$.

Suppose $y^1, \ldots, y^k : \mathsf{H}(n)$ are H-vars defined by $y^i := \phi^i(g^1, \ldots, g^M; \Theta^i)$ for (possibly self-)parametrized nonlinearities $\phi^i(-;-) : \mathbb{R}^m \to \mathbb{R}, i \in [k]$ and parameters Θ^i (possibly dependent on previous G-vars). Suppose that each Θ^i converges almost surely to a deterministic vector $\mathring{\Theta}^i$, and suppose each ϕ^i is parameter-controlled at $\mathring{\Theta}^i$. Each of y^i has mean

$$\nu(y^i) \overset{\text{def}}{=} \frac{1}{n}\sum_{\alpha=1}^n y_\alpha^i = \frac{1}{n}\sum_{\alpha=1}^n \phi^i(g_\alpha^1, \ldots, g_\alpha^M; \Theta^i)$$

and variance

$$\sigma^2(y^i) \overset{\text{def}}{=} \frac{1}{n}\sum_{\alpha=1}^n (y_\alpha^i)^2 - \nu(y^i)^2 = \frac{1}{n}\sum_{\alpha=1}^n \phi^i(g_\alpha^1, \ldots, g_\alpha^M; \Theta^i)^2 - \nu(y^i)^2.$$

Under generic conditions (i.e. Assumption C.3 and parameter-control), Theorem C.4 or Theorem C.11 applies, so that

$$\nu(y^i) \xrightarrow{\text{a.s.}} \mathring{\nu}(y^i) \overset{\text{def}}{=} \mathop{\mathbb{E}}_Z \phi^i\left(Z; \mathring{\Theta}^i\right), \text{ and } \sigma^2(y^i) \xrightarrow{\text{a.s.}} \mathring{\sigma}^2(y^i) \overset{\text{def}}{=} \mathop{\mathbb{E}}_Z \phi^i\left(Z; \mathring{\Theta}^i\right)^2 - \left[\mathop{\mathbb{E}}_Z \phi^i\left(Z; \mathring{\Theta}^i\right)\right]^2$$

where $Z \sim \mathcal{N}(\mu, \Sigma)$. Layernorm$(y^i)$ can then be expressed via a self-parametrized (Definition C.8) `Nonlin`$^+$ rule like so

$$\text{Layernorm}(y^i) = \psi(y^i; \nu(y^i), \sigma^2(y^i)), \quad \text{where} \quad \psi(z; a, b) \overset{\text{def}}{=} (z-a)/\sqrt{b}.$$

It's easy to check that $\psi(z; a, b)$ is parameter-controlled at $\mathring{a}, \mathring{b}$ as long as $\mathring{b} \neq 0$. Assuming rank stability (Assumption C.3) is not violated by the new variables, Theorem C.4 holds, so that, intuitively, this application of `Nonlin`$^+$ can be replaced with a straightforward application of `Nonlin`:

$$\text{“Layernorm}(y^i) = \psi(y^i; \nu(y^i), \sigma^2(y^i))\text{”} \to \text{“Layernorm}(y^i) = \psi(y^i; \mathring{\nu}(y^i), \mathring{\sigma}^2(y^i))\text{”}.$$

Therefore, if we define the kernel matrices

$$\Omega_{ij} = \lim_{n\to\infty} y^{i\top} y^j/n$$
$$\bar{\Omega}_{ij} = \lim_{n\to\infty} \text{Layernorm}(y^i)^\top \text{Layernorm}(y^j)/n,$$

then

$$\begin{aligned}
\bar{\Omega}_{ij} &= \lim_{n\to\infty} \frac{1}{n} \frac{(y_i - \nu(y^i))^\top (y_j - \nu(y^j))}{\sqrt{\sigma^2(y^i)\sigma^2(y^j)}} \\
&= \lim_{n\to\infty} \frac{y_i^\top y_j/n - \nu(y^i)\nu(y^j)}{\sqrt{\sigma^2(y^i)\sigma^2(y^j)}} \\
&= \lim_{n\to\infty} \frac{y_i^\top y_j/n - \mathring{\nu}(y^i)\mathring{\nu}(y^j)}{\sqrt{\mathring{\sigma}^2(y^i)\mathring{\sigma}^2(y^j)}} \\
\bar{\Omega} &= D^{-1/2}(\Omega - \mathring{\nu}\mathring{\nu}^\top)D^{-1/2},
\end{aligned}$$

where $\mathring{\nu}$ is the column vector $(\mathring{\nu}(y^1), \ldots, \mathring{\nu}(y^k))^\top$ and $D = \mathrm{Diag}(\Omega - \mathring{\nu}\mathring{\nu}^\top)$.

In summary,

Computing Layernorm Kernel

Suppose $y^1, \ldots, y^k : \mathsf{H}(n)$ are H-vars defined by $y^i := \phi^i(g^1, \ldots, g^M; \Theta^i)$ for (possibly self-)parametrized nonlinearities $\phi^i(-;-) : \mathbb{R}^m \to \mathbb{R}, i \in [k]$ and parameters Θ^i. Assume that each Θ^i converges almost surely to a deterministic vector $\mathring{\Theta}^i$, and that each ϕ^i is parameter-controlled at $\mathring{\Theta}^i$. If we define the kernel matrices

$$\begin{aligned}
\Omega_{ij} &= \lim_{n\to\infty} {y^i}^\top y^j/n \\
\bar{\Omega}_{ij} &= \lim_{n\to\infty} \mathrm{Layernorm}(y^i)^\top \mathrm{Layernorm}(y^j)/n,
\end{aligned}$$

then, assuming generic conditions (see main text above),

$$\bar{\Omega} = D^{-1/2}(\Omega - \mathring{\nu}\mathring{\nu}^\top)D^{-1/2},$$

where $D = \mathrm{Diag}(\Omega - \mathring{\nu}\mathring{\nu}^\top)$ and $\mathring{\nu}$ is the vector given by $\mathring{\nu}_i = \mathbb{E}\,\phi^i(Z; \mathring{\Theta}^i), Z \sim \mathcal{N}(\mu, \Sigma)$.

D.3 Transformer (Program 10)

The Transformer Variant, in Mathematical Terms We'll work with the following transformer model. Let $x_1^0, \ldots, x_t^0$ be a sequence of inputs (the superscript will be layer index, and the subscript will be token index). Then each layer l of our transformer works like the following

$$\begin{aligned}
k_i^l &= U^l x_i^{l-1} \in \mathbb{R}^n \\
h_i^l &= \mathrm{Layernorm}(k_i^l + \mathrm{MaskedAttention}_i(k_i^l, \{k_j^l\}_{j=1}^t, \{k_j^l\}_{j=1}^t)) \\
x_i^l &= \mathrm{Layernorm}(W^{l2}\mathrm{relu}(W^{l1}h_i^l + b^{l1}) + b^{l2} + W^{l1}h_i^l) \qquad (42)
\end{aligned}$$

where U^l, W^{l1}, W^{l2} are weights and b^{l1}, b^{l2} are the biases, and

$$\begin{aligned}
\mathrm{MaskedAttention}_j(q, \{k^i\}_{i=1}^r, \{v^i\}_{i=1}^r) &= \sum_{i=1}^r a_i^j v^i, \\
\text{where} \quad a_i^j &= \mathrm{SoftMax}(q^\top k^1/n, \ldots, q^\top k^j/n, -\infty, \ldots, -\infty)_i \qquad (43)
\end{aligned}$$

as described in Appendix A.

Note that we make the following simplifications for ease of presentation, but all of them can be removed at the expense of more complex NETSOR programs.

1. We are forgoing positional embeddings
2. The keys, values, and queries here are the same, compared to the standard version, where they are different linear projections of x_i^{l-1}
3. There is only 1 head, compared to the standard multi-head attention
4. The skip connection has base $W^{l2}h_i^l$ instead of just h_i^l

Self-parametrized Netsor$^+$ program 10 Transformer

```
Input: $U^1 x_1^0, \dots, U^1 x_t^0 : \mathsf{G}(n)$
Input: $\forall l = 1, \dots, L : W^{l1}, W^{l2} : \mathsf{A}(n, n)$
Input: $\forall l = 2, \dots, L : U^l : \mathsf{A}(n, n)$
Input: $\forall l = 1, \dots, L : b^{l1}, b^{l2} : \mathsf{G}(n)$
Input: $v : \mathsf{G}(n)$
 1: for $l = 1, \dots, L$ do
 2:     for $i = 1, \dots, t$ do
 3:         // if $l = 1$, apply LinComb
 4:         // if $l \geq 2$, apply MatMul
 5:         $k_i^l := U^l x_i^{l-1} : \mathsf{G}(n)$
 6:     end for
 7:     for $i = 1, \dots, t$ do
 8:         for $j = 1, \dots, t$ do
 9:             // Moment
10:             $c_{ij} := k_i^{l\top} k_j^l / n : \mathsf{C}$
11:         end for
12:         // With $a_j^i$ being shorthand for
13:         // $\mathrm{SoftMax}(c_{i1}, \dots, c_{ii}, -\infty, \dots, -\infty)_j$
14:         // Mean, post attention
15:         $\nu_i := \frac{1}{n} \sum_{\alpha=1}^n (k_i^l + \sum_{j=1}^t a_j^i k_j^l)_\alpha : \mathsf{C}$
16:         // Variance, post attention
17:         $var_i = \frac{1}{n} \sum_{\alpha=1}^n (k_i^l + \sum_{j=1}^t a_j^i k_j^l)_\alpha^2 - \nu_i^2 : \mathsf{C}$
18:         // applying Nonlin$^+$ to express attention+layernorm
19:         $h_i^l := (k_i^l + \sum_{j=1}^t a_j^i k_j^l - \nu_i \mathbf{1}) / \sqrt{var_i} : \mathsf{H}(n)$
20:     end for
21:     for $i = 1, \dots, t$ do
22:         $y_i^{l1} := W^{l1} h_i^l : \mathsf{G}(n)$
23:         $\hat{y}_i^{l1} := y_i^{l1} + b^{l1} : \mathsf{G}(n)$
24:         $\hat{x}_i^{l1} := \mathrm{ReLU}(\hat{y}_i^{l1}) : \mathsf{H}(n)$
25:         $y_i^{l2} := W^{l2} \hat{x}_i^{l1} : \mathsf{G}(n)$
26:         $\hat{y}_i^{l2} := y_i^{l2} + b^{l2} : \mathsf{G}(n)$
27:         // Layernorm mean and variance
28:         $\nu_i' := \frac{1}{n} \sum_{\alpha=1}^n (\hat{y}_i^{l2})_\alpha + (y_i^{l1})_\alpha : \mathsf{C}$
29:         $var_i' := \frac{1}{n} \sum_{\alpha=1}^n ((\hat{y}_i^{l2})_\alpha + (y_i^{l1})_\alpha)^2 - (\nu_i')^2 : \mathsf{C}$
30:         // Layernorm
31:         $x_i^l := (\hat{y}_i^{l2} + y_i^{l1} - \nu_i' \mathbf{1}) / \sqrt{var_i'} : \mathsf{H}(n)$
32:     end for
33: end for
Output: $(v^\top x_1^L / \sqrt{n}, \dots, v^\top x_t^L / \sqrt{n})$
```

Setup assume for all $\alpha, \beta \in [n]$,

- $W_{\alpha\beta}^{l1}, W_{\alpha\beta}^{l2} \sim \mathcal{N}(0, \sigma_w^2/n)$ for all $l \geq 1$
- $U_{\alpha\beta}^l \sim \mathcal{N}(0, \sigma_u^2/n)$ for all $l \geq 2$ and $U_{\alpha\beta}^1 \sim \mathcal{N}(0, \sigma_u^2/m)$
- $b_\alpha^{l1}, b_\alpha^{l2} \sim \mathcal{N}(0, \sigma_b^2)$ for all l.
- $v_\alpha \sim \mathcal{N}(0, \sigma_v^2)$

D.3.1 Expressing the Composition of Attention, Skip Connection, and Layernorm via `Nonlin`$^+$ and `Moment`

Program 10 captures the computation of this transformer on an input sequence. Let us explain how Eq. (42) is expressed in Program 10. Throughout the below, we will use the easy observation that $\mu(g) = 0$ for all G-vars g. For any layer l, we proceed as follows.

Attention Weights First, c_{ij} in Line 10 represents a pre-SoftMax logit for the attention weights. They are introduced via `Moment` by

$$c_{ij} := \frac{1}{n}\sum_{\alpha=1}^{n} \varphi^{c_{ij}}((k_i^l)_\alpha, (k_j^l)_\alpha), \quad \text{where } \varphi^{c_{ij}}(z_1, z_2) = z_1 z_2.$$

This implies

$$\mathring{c}_{ij} = \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} Z^{k_i^l} Z^{k_j^l} = \Sigma(k_i^l, k_j^l), \tag{44}$$

where we used $\mu(g) = 0$ for all G-vars g.

Layernorm Mean and Variance Next, ν_i in Line 15 and var_i in Line 17 represent the mean and variance of the post-attention embedding of the ith token. They are introduced via `Moment` by

$$\nu_i := \frac{1}{n}\sum_{\alpha=1}^{n} \varphi^{\nu_i}((k_1^l)_\alpha, \dots, (k_t^l)_\alpha; c_{i1}, \dots, c_{ii})$$
$$var_i := \frac{1}{n}\sum_{\alpha=1}^{n} \varphi^{var_i}((k_1^l)_\alpha, \dots, (k_t^l)_\alpha; c_{i1}, \dots, c_{ii}, \nu_i)$$

where

$$\varphi^{\nu_i}(z_1, \dots, z_t; \theta_1, \dots, \theta_i) \overset{\text{def}}{=} z_i + \sum_{j=1}^{t} a_j z_j,$$
$$\text{where } (a_1, \dots, a_t) = \mathrm{SoftMax}(\theta_1, \dots, \theta_i, -\infty, \dots, -\infty), \tag{45}$$

and similarly,

$$\varphi^{var_i}(z_1, \dots, z_t; \theta_1, \dots, \theta_i, \nu) \overset{\text{def}}{=} (z_i + \sum_{j=1}^{t} a_j z_j)^2 - \nu^2,$$
$$\text{where } (a_1, \dots, a_t) \text{ are as in Eq. (45).}$$

Note that both φ^{ν_i} and φ^{var_i} are pseudo-Lipschitz in both their inputs and parameters jointly, so that they are parameter-controlled by Example C.2.

Their limit parameters can be computed as

$$\mathring{\nu}_i = \mu(k_i^l) + \sum_{j=1}^{t} \mathring{a}_j \mu(k_j^l) = 0$$
$$\text{where } (\mathring{a}_1, \dots, \mathring{a}_t) = \mathrm{SoftMax}(\mathring{\theta}_1, \dots, \mathring{\theta}_i, -\infty, \dots, -\infty), \tag{46}$$

since $\mu = 0$ identically, and

$$v\mathring{a}r_i = \Sigma(k_i^l, k_i^l) + 2\sum_j \mathring{a}_j \Sigma(k_i^l, k_j^l) + \sum_{j,j'} \mathring{a}_j \mathring{a}_{j'} \Sigma(k_j^l, k_{j'}^l) \tag{47}$$

with $\mathring{a}_j$ same as in Eq. (46).

Putting Them All Together Finally, h_i^l in Line 19 represents the post-layernorm activations and is introduced via `Nonlin`$^+$ by

$$h_i^l := \varphi^{h_i^l}(k_1^l, \dots, k_t^l; c_{i1}, \dots, c_{ii}, \nu_i, var_i)$$

where

$$\varphi^{h_i^l}(z_1, \dots, z_t; \theta_1, \dots, \theta_i, \nu, var) := (z_i + \sum_{j=1}^{t} a_j z_j - \nu)/\sqrt{var} \tag{48}$$
$$\text{where } (a_1, \dots, a_t) \text{ are as in Eq. (45).}$$

If $v\mathring{a}r_i > 0$, then one can show that $\varphi^{h_i^l}$ is parameter-controlled at $(\mathring{c}_{i1}, \dots, \mathring{c}_{ii}, \mathring{\nu}_i, v\mathring{a}r_i)$ via the same reasoning as in Appendix D. When is $v\mathring{a}r_i > 0$? From Eq. (47), because the a_i are all nonnegative, $v\mathring{a}r_i = 0$ implies that $\Sigma(k_i^l, k_i^l) = 0$. This is impossible if all of the input tokens x_i are nonzero and the weight variances satisfy $\sigma_w, \sigma_u > 0$, as one can easily see.

D.3.2 Computing the GP Kernel

By Corollary C.13, the output vector converges in distribution to $\mathcal{N}(0, K)$, where $K \in \mathbb{R}^{t\times t}$, and

$$K_{ij} = \sigma_v^2 \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \varphi^{x_i^L}(Z; \mathring{\Theta}^{x_i^L})\varphi^{x_j^L}(Z; \mathring{\Theta}^{x_j^L}).$$

Here $\Theta^{x_i^L} = \{\nu_i', var_i'\}$ as given in Lines 28 and 29, and

$$\varphi^{x_i^L}(Z; \nu, var) = (Z^{\hat{y}_i^{L2}} + Z^{y_i^{l1}} - \nu)/\sqrt{var}.$$

Simultaneously,

$$\mathring{\nu}_i' = \mathop{\mathbb{E}}_{Z} Z^{\hat{y}_i^{L2}} + Z^{y_i^{L1}}, \quad \mathring{var}_i' = \mathop{\mathbb{E}}_{Z}(Z^{\hat{y}_i^{L2}} + Z^{y_i^{L1}})^2 - (\mathring{\nu}_i')^2.$$

Thus, to compute K, it suffices to compute the restriction $\Sigma|_{y_1^{L1},\ldots,y_t^{L1},\hat{y}_1^{L2},\ldots,\hat{y}_t^{L2}}$, from which K can be computed by the equations above.

However, notice that by the "otherwise" case Eq. (37), $\Sigma(h_i^{L2}, h_i^{L1}) = 0$ because $\hat{h}_i^{L2}$ and h_i^{L1} are introduced by `MatMul` with different A-vars, and consequently $\Sigma(\hat{h}_i^{L2}, h_i^{L1}) = \Sigma(h_i^{L2}, h_i^{L1}) + \Sigma(b^{l2}, h_i^{L1}) = 0$. Therefore, we only need to compute $\Sigma|_{y_1^{L1},\ldots,y_t^{L1}}$ and $\Sigma|_{\hat{y}_1^{L2},\ldots,\hat{y}_t^{L2}}$ separately. Then K is given by

$$K = \sigma_v^2 D^{-1/2}(\Sigma|_{y_1^{L1},\ldots,y_t^{L1}} + \Sigma|_{\hat{y}_1^{L2},\ldots,\hat{y}_t^{L2}})D^{-1/2}, \tag{49}$$

where D is the diagonal matrix with diagonal equal to the diagonal of $\Sigma|_{y_1^{L1},\ldots,y_t^{L1}} + \Sigma|_{\hat{y}_1^{L2},\ldots,\hat{y}_t^{L2}}$.

D.3.3 Computing Σ

Let

$$\Sigma^{\hat{y}^{l2}} \overset{\text{def}}{=} \Sigma|_{\hat{y}_1^{l2},\ldots,\hat{y}_t^{l2}}, \quad \Sigma^{y^{l1}} \overset{\text{def}}{=} \Sigma|_{y_1^{l1},\ldots,y_t^{l1}}, \quad \Sigma^{k^l} \overset{\text{def}}{=} \Sigma|_{k_1^l,\ldots,k_t^l}$$

resp. be the restriction of Σ to $\{\hat{y}_i^{l2}\}_i$, $\{y_i^{l1}\}_i$, and $\{\hat{k}_i^l\}_i$. As explained above, the kernel of the Gaussian process underlying the output vector $(v^\top x_1^L/\sqrt{n}, \ldots, v^\top x_t^L/\sqrt{n})$ can be computed from $\Sigma^{\hat{y}^{L2}}$.

In this section, we shall describe equations tying together $\Sigma^{\hat{y}^{l2}}$, $\Sigma^{y^{l1}}$, Σ^{k^l} that will allow us to compute $\Sigma^{\hat{y}^{L2}}$ recursively.

Computing $\Sigma^{y^{l1}}$ from Σ^{k^l}. The G-var y_i^{l1} is introduced as $y_i^{l1} := W^{l1}h_i^l$. Then given Eq. (48), we have, for any $i, i' \in [t]$,

$$\Sigma(y_i^{l1}, y_{i'}^{l1}) = \frac{\sigma_w^2}{\sqrt{\mathring{var}_i \mathring{var}_{i'}}}\left(\Sigma(k_i^l, k_{i'}^l) + \sum_j \mathring{a}_j^i \Sigma(k_j^l, k_{i'}^l) + \sum_{j'} \mathring{a}_{j'}^{i'}\Sigma(k_i^l, k_{j'}^l) + \sum_{j,j'} \mathring{a}_j^i \mathring{a}_{j'}^{i'}\Sigma(k_j^l, k_{j'}^l)\right), \tag{50}$$

where

$$\begin{aligned}(\mathring{a}_1^i, \ldots, \mathring{a}_t^i) &= \mathrm{SoftMax}(\mathring{c}_{i1}, \ldots, \mathring{c}_{ii}, -\infty, \ldots, -\infty)\\ &= \mathrm{SoftMax}(\Sigma(k_i^l, k_1^l), \ldots, \Sigma(k_i^l, k_i^l), -\infty, \ldots, -\infty)\end{aligned}$$

by Eq. (44), and likewise for i'. This reduces computing $\Sigma^{y^{l1}}$ to computing Σ^{k^l}.

Computing $\Sigma^{\hat{y}^{l2}}$ from $\Sigma^{y^{l1}}$. By some simple calculations in the vein of Appendix B.1.2, we can also see

$$\Sigma^{\hat{y}^{l2}} = \sigma_w^2 \mathrm{V}_{\mathrm{ReLU}}\left(\Sigma^{y^{l1}} + \sigma_b^2\right) + \sigma_b^2. \tag{51}$$

Computing $\Sigma^{k^{l+1}}$ from $\Sigma^{\hat{y}^{l2}}$. Finally, following the same reasoning as in Appendix D.1, we get

$$\mathring{\nu}'_i = 0, \quad \mathring{var}'_i = \Sigma(\hat{y}^{l2}_i, \hat{y}^{l2}_i) + \Sigma(y^{l1}_i, y^{l1}_i),$$

$\varphi^{x^l_i}$ is parameter-controlled at $\mathring{\Theta}^{x^l_i}$ as long as $\mathring{var}'_i > 0$, and

$$\Sigma^{k^{l+1}} = \sigma_u^2 D^{-1/2}(\Sigma^{\hat{y}^{l2}} + \Sigma^{y^{l1}})D^{-1/2} \tag{52}$$

where $D = \mathrm{Diag}(\Sigma^{\hat{y}^{l2}} + \Sigma^{y^{l1}})$.

Putting them all together, Eqs. (50) to (52) along with Eq. (49) yield the complete set of equations to compute the GP kernel of a transformer.

D.3.4 Vectorized Implementation: Single Sequence

Eqs. (49), (51) and (52) are already in vectorized forms. The following equation expresses Eq. (50) in a vectorized form as well:

$$\Sigma^{y^l} = \sigma_w^2 D^{-1/2}(I+\Delta)\Sigma^{k^l}(I+\Delta)^\top D^{-1/2}$$

where

- $\Delta = \mathrm{SoftMax}(\mathrm{Mask}(\Sigma^{k^l}))$, with SoftMax applied to each row, and $\mathrm{Mask}(\Sigma^{k^l})$ is the same as Σ^{k^l}, except that its upper triangular portion (above the diagonal) is all set to $-\infty$, and
- D is the diagonal matrix with diagonal equal to the diagonal of $(I+\Delta)\Sigma^{k^l}(I+\Delta)^\top$.

Here, Δ is the attention weights, masked so that a token's embedding cannot depend on those of future tokens. The identity matrix I appears due to the skip connection. And the multiplication by $D^{-1/2}$ is as result of layernorm.

D.3.5 Vectorized Implementation: Double Sequence

Program 10 only expresses the computation of a transformer on a single sequence. In general, the GP kernel will also have covariances between the embeddings of tokens of one sequence and those of tokens of another sequence. One can derive the computation of these covariances just as we did above for a single sequence. Below, we will just summarize the vectorized implementation for computing the joint kernel over multiple input sequences. One should think of $\mathbf{\Sigma}^l$ below as the tensor of Σ^{k^l} over every pair of sequences, and one should think of $\hat{\mathbf{\Sigma}}^l$ as the same for Σ^{y^l}.

Computing Transformer Kernel

Suppose we have p input sequences $\{(x_{1a}, \dots, x_{ta})\}_{a=1}^{p}$, each with t tokens. Suppose each sequence is processed by a transformer as in Program 10, and the transformer's parameters are sampled with nonzero variances as follows.

- $W^{l1}_{\alpha\beta}, W^{l2}_{\alpha\beta} \sim \mathcal{N}(0, \sigma_w^2/n)$ for all $l \geq 1$
- $U^{l}_{\alpha\beta} \sim \mathcal{N}(0, \sigma_u^2/n)$ for all $l \geq 2$ and $U^{1}_{\alpha\beta} \sim \mathcal{N}(0, \sigma_u^2/m)$
- $b^{l1}_{\alpha}, b^{l2}_{\alpha} \sim \mathcal{N}(0, \sigma_b^2)$ for all l.
- $v_\alpha \sim \mathcal{N}(0, \sigma_v^2)$

Then the transformer's outputs, one scalar for each input token, converge in distribution to a Gaussian $\mathcal{N}(0, K)$ where $K \in \mathbb{R}^{pt \times pt}$ can be computed as follows:

```
1. Initialize $\mathbf{\Sigma}^0 \in \mathbb{R}^{t\times p\times t\times p}$ by $\mathbf{\Sigma}^0_{iajb} \leftarrow \sigma_u^2 x_{ia}^\top x_{jb}/m$ for all $a, b \in [p]$ and $i, j \in t$.
2. For $l = 1, \dots, L$, do
   (a) For $a = 1, \dots, p$, do
       i. $\Sigma^{l-1,a} \leftarrow \mathbf{\Sigma}^{l-1}_{:a:a}$.
       ii. $\Delta^{la} \leftarrow \mathrm{SoftMax}(\mathrm{Mask}(\Sigma^{l-1,a}))$, where Mask replaces the upper triangular portion (above the diagonal) with $-\infty$, and SoftMax is applied row-wise.
   (b) $\mathbf{\Delta}^l \leftarrow$ block diagonal matrix with $\Delta^{l1}, \dots, \Delta^{lp}$ as blocks.
   (c) // below, we treat each tensor as a $(pt \times pt)$ matrix.
   (d) $\hat{\mathbf{\Sigma}}^l \leftarrow (I + \mathbf{\Delta}^l)\mathbf{\Sigma}^{l-1,a}(I + \mathbf{\Delta}^l)^\top$
   (e) $\hat{\mathbf{\Sigma}}^l \leftarrow \sigma_w^2 D^{-1/2}\hat{\mathbf{\Sigma}}^l D^{-1/2}$, where $D = \mathrm{Diag}(\mathbf{\Sigma}^l)$
   (f) $\mathbf{\Sigma}^l \leftarrow \sigma_w^2 \mathrm{V}_{\mathrm{ReLU}}(\hat{\mathbf{\Sigma}}^l + \sigma_b^2) + \sigma_b^2$
   (g) $\mathbf{\Sigma}^l \leftarrow \sigma_u^2 D^{-1/2}(\mathbf{\Sigma}^l + \hat{\mathbf{\Sigma}}^l)D^{-1/2}$, where $D = \mathrm{Diag}(\mathbf{\Sigma}^l + \hat{\mathbf{\Sigma}}^l)$
3. Return $\frac{\sigma_v^2}{\sigma_u^2}\mathbf{\Sigma}^L$
```

See our repo `github.com/thegregyang/GP4A` for an implementation of this algorithm.

E Different Versions of Tensor Programs

Definition E.1. A NETSOR$^-$ program is a NETSOR program without the `LinComb` rule.

Remark E.2. Any NETSOR program is semantically identical to a NETSOR$^-$ program, by absorbing any usage of `LinComb` into a downstream nonlinearity (e.g., if $g := g^1 + g^2$, and $h := \phi(g)$, write $h := \phi(g^1 + g^2)$ directly as an application of `Nonlin`), or if there is no downstream nonlinearity, treat it as an application of `Nonlin`. Because `LinComb` allows one to express certain gadgets such as skip connection and convolutions more easily, we chose to present NETSOR as the canonical version of Tensor Program here. See Appendix J for a formal specification of NETSOR$^-$.

By the remark above, the following NETSOR$^-$ Master Theorem is equivalent to Theorem 5.4.

Theorem E.3 (NETSOR$^-$ Master Theorem)**.** *Fix any NETSOR$^-$ program satisfying Assumption 5.1 and with all nonlinearities controlled. If $g^1, \dots, g^M$ are all of the G-vars (including all input G-vars), then for any controlled $\psi : \mathbb{R}^M \to \mathbb{R}$, as $n \to \infty$,*

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g^1_\alpha, \dots, g^M_\alpha) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z) = \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \psi(Z^{g^1}, \dots, Z^{g^M}),$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence, $Z = (Z^{g^1}, \dots, Z^{g^M}) \in \mathbb{R}^M$, and $\mu = \{\mu(g^i)\}_{i=1}^M \in \mathbb{R}^M$ and $\Sigma = \{\Sigma(g^i, g^j)\}_{i,j=1}^M \in \mathbb{R}^{M\times M}$ are given in Eq. (2) (note that the cases involving `LinComb` in Eq. (2) are now vacuous in this setting with NETSOR$^-$ program). See Fig. 1 for an illustration.

To prove Theorem 5.4, we will in fact prove Theorem E.3; see Appendix H.

Definition E.4. A NETSOR$\circ$ program (pronounced "Net-Sor-O") is a NETSOR program but where Nonlin rules allow nonlinearities ϕ to take H-vars. NETSOR$\circ$ programs are thus a superset of NETSOR programs. Similarly, a NETSOR$\circ^+$ (pronounced "Net-Sor-O-Plus") program is a NETSOR$^+$ program but where Nonlin$^+$ rules allow nonlinearities ϕ to take H-vars.

Remark E.5. Any NETSOR$\circ$ program is semantically identical to a NETSOR program: If $g := Wh$ is any application of MatMul, we can rewrite h as a function of G-vars only by unwinding its definition recursively (e.g., if $h := \phi(h^1, g)$ and $h^1 := \psi(g^1, g^2)$, then we can write directly $h := \phi(\psi(g^1, g^2), g)$ using a single application of Nonlin in G-vars). Likewise, any NETSOR$\circ^+$ program can be rewritten as a NETSOR$^+$ program without losing any information.

NETSOR$\circ$ programs can be more concise than NETSOR programs by reusing H-vars more efficiently; see Program 11 for GRU expressed in NETSOR$\circ$, and compare to Program 5. However, the Master Theorem is more complicated to state, and the task of unwinding the nonlinearity just shifts from the program to the scaling limit computation stage; see Eq. (53) below. This is why we did not present NETSOR$\circ$ as the canonical version of Tensor Programs.

Definition E.6. Fix a NETSOR$\circ$ program. For any H-var h, let φ^h be the unwinded nonlinearity expressing h as a function of only G-vars, as described in Remark E.5, i.e. $h = \varphi^h(g^1, \dots, g^M)$. For example, if $h := \phi(h^1, g^3)$ and $h^1 := \psi(g^1, g^2)$, then $h = \phi(\psi(g^1, g^2), g^3)$ and $\varphi^h = \phi(\psi(-,-),-)$.

Similarly, in a NETSOR$\circ^+$ program, if h is an H-var, let φ^h be the unwinded nonlinearity (possibly with parameters) expressing h as a function only G-vars, $h = \varphi^h(g^1, \dots, g^M; \Theta)$.

For example, if $h := \phi(h^1, g^3; \theta_2)$ and $h^1 := \psi(g^1, g^2; \theta_1)$, then $h = \phi(\psi(g^1, g^2; \theta_1), g^3; \theta_2)$ and $\varphi^h(-,-,-;\theta_1,\theta_2) = \phi(\psi(-,-;\theta_1),-;\theta_2)$.

Note that this φ^h notation is consistent with the semantics of the same notation defined in Definition C.10, where there is nothing to unwind.

The extended mean and covariance μ and Σ can still be computed as before in a NETSOR$\circ$ program. The only difference is that we are using the unwinded nonlinearities φ^h instead.

$$\mu(g) = \begin{cases} \mu^{\text{in}}(g) & \text{if } g \text{ is input} \\ \sum_i a_i \mu(y^i) & \text{if } g = \sum_i a_i y^i \text{, introduced by LinComb ,} \\ 0 & \text{otherwise} \end{cases}$$

$$\Sigma(g, g') = \begin{cases} \Sigma^{\text{in}}(g, g') & \text{if } g, g' \text{ are inputs} \\ \sum_i a_i \Sigma(y^i, g') & \text{if } g = \sum_i a_i y^i \text{, introduced by LinComb} \\ \sum_i a_i \Sigma(g, y^i) & \text{if } g' = \sum_i a_i y^i \text{, introduced by LinComb} \\ \sigma_W^2 \mathbb{E}_Z \varphi^h(Z) \varphi^{h'}(Z) & \text{if } g = Wh, g' = Wh' \text{, introduced by MatMul w/ same A-var } W \\ 0 & \text{otherwise} \end{cases} \tag{53}$$

where φ^h and $\varphi^{h'}$ is as defined in Definition E.6 and $Z \sim \mathcal{N}(\mu, \Sigma)$.

Theorem E.7 (NETSOR$\circ$ Master Theorem)**.** *Fix any* NETSOR$\circ$ *program satisfying Assumption 5.1 and with all unwinded nonlinearities φ^h controlled, for all H-vars h. If $g^1, \dots, g^M$ are all of the G-vars (including all input G-vars), then for any controlled $\psi : \mathbb{R}^M \to \mathbb{R}$, as $n \to \infty$,*

$$\frac{1}{n} \sum_{\alpha=1}^{n} \psi(g^1_\alpha, \dots, g^M_\alpha) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z) = \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z^{g^1}, \dots, Z^{g^M}),$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence, $Z = (Z^{g^1}, \dots, Z^{g^M}) \in \mathbb{R}^M$, and $\mu = \{\mu(g^i)\}_{i=1}^M \in \mathbb{R}^M$ and $\Sigma = \{\Sigma(g^i, g^j)\}_{i,j=1}^M \in \mathbb{R}^{M \times M}$ are given in Eq. (53). See Fig. 1 for an illustration.

Theorem E.8. *Fix any* NETSOR$\circ^+$ *program satisfying Assumption 5.1 and Assumption C.3. Suppose for each parametrized unwinded nonlinearity $\varphi^h(-;\Theta)$, the parameters Θ are instantiated with random variables that converge almost surely to some deterministic vector $\mathring{\Theta}$ as $n \to \infty$, and assume φ^h is parameter-controlled at $\mathring{\Theta}$. If $g^1, \dots, g^M$ are all of the G-vars (including all input G-vars), then for any l, for any random vector $\Theta \in \mathbb{R}^l$ that converges almost surely to a deterministic vector*

NETSOR∘ program 11 GRU, with Gating Function σ and Activation Function ϕ

```
    // Embeddings of input sequence
Input: $U_{\mathsf{z}}x^1, \ldots, U_{\mathsf{z}}x^t : \mathsf{G}(n)$
Input: $U_{\mathsf{r}}x^1, \ldots, U_{\mathsf{r}}x^t : \mathsf{G}(n)$
Input: $U_{\mathsf{h}}x^1, \ldots, U_{\mathsf{h}}x^t : \mathsf{G}(n)$
    // Parameters
Input: $W_{\mathsf{z}}, W_{\mathsf{r}}, W_{\mathsf{h}} : \mathsf{A}(n, n)$
Input: $b_{\mathsf{z}}, b_{\mathsf{r}}, b_{\mathsf{h}} : \mathsf{G}(n)$
    // Initial GRU state
Input: $h^0 : \mathsf{G}(n)$
    // Readout layer
Input: $v : \mathsf{G}(n)$
    // Time step 1
    $h_{\mathsf{z}}^1 := W_{\mathsf{z}}h^0 : \mathsf{G}(n)$
    $\tilde{z}^1 := h_{\mathsf{z}}^1 + U_{\mathsf{z}}x^1 + b_{\mathsf{z}} : \mathsf{G}(n)$
    $h_{\mathsf{r}}^1 := W_{\mathsf{r}}h^0 : \mathsf{G}(n)$
    $\tilde{r}^1 := h_{\mathsf{r}}^1 + U_{\mathsf{r}}x^1 + b_{\mathsf{r}} : \mathsf{G}(n)$
    // $\sigma$ is gating function, typically sigmoid; applying Nonlin
    $\hat{h}^0 := h^0 \odot \sigma(\tilde{r}^1) : \mathsf{H}(n)$
    $h_{\mathsf{h}}^1 := W_{\mathsf{h}}\hat{h}^0 : \mathsf{G}(n)$
    $\tilde{h}^1 := h_{\mathsf{h}}^1 + U_{\mathsf{h}}x^1 + b_{\mathsf{h}} : \mathsf{G}(n)$
    // Apply Nonlin
    // $\phi$ is activation function, typically $\tanh$
    $h^1 := (1 - \sigma(\tilde{z}^1)) \odot h^0 + \sigma(\tilde{z}^1) \odot \phi(\tilde{h}^1) : \mathsf{H}(n)$
    // Time step 2
    $h_{\mathsf{z}}^2 := W_{\mathsf{z}}h^1 : \mathsf{G}(n)$
    $\tilde{z}^2 := h_{\mathsf{z}}^2 + U_{\mathsf{z}}x^2 + b_{\mathsf{z}} : \mathsf{G}(n)$
    $h_{\mathsf{r}}^2 := W_{\mathsf{r}}h^1 : \mathsf{G}(n)$
    $\tilde{r}^2 := h_{\mathsf{r}}^2 + U_{\mathsf{r}}x^2 + b_{\mathsf{r}} : \mathsf{G}(n)$
    // No longer need to unwind $h^1$ as in Program 5
    $\hat{h}^1 = \sigma(\tilde{r}^1) \odot h^1 : \mathsf{H}(n)$
    $h_{\mathsf{h}}^2 := W_{\mathsf{h}}\hat{h}^1 : \mathsf{G}(n)$
    $\tilde{h}^2 := h_{\mathsf{h}}^2 + U_{\mathsf{h}}x^2 + b_{\mathsf{h}} : \mathsf{G}(n)$
    // No longer need to unwind $h^1$ as in Program 5
    $h^2 := (1 - \sigma(\tilde{z}^2)) \odot h^1 + \sigma(\tilde{z}^2) \odot \phi(\tilde{h}^2) : \mathsf{H}(n)$
    // Time step 3
    ⋮
    // Time step t
    // Define $\tilde{z}^t, \tilde{r}^t, \tilde{h}^t$ just like above
    ⋮
    // No longer need to unwind $ht - 1$ as in Program 5
    $h^t := (1 - \sigma(\tilde{z}^t)) \odot h^{t-1} + \sigma(\tilde{z}^t) \odot \phi(\tilde{h}^t) : \mathsf{H}(n)$
Output: $(v^\top h^1/\sqrt{n}, \ldots, v^\top h^t/\sqrt{n})$
```

$\mathring{\Theta}$, as $n \to \infty$, and for any $\psi : \mathbb{R}^M \times \mathbb{R}^l \to \mathbb{R}$ parameter-controlled at $\mathring{\Theta}$,

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g_\alpha^1, \ldots, g_\alpha^M; \Theta) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z; \mathring{\Theta}),$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence, $Z \in \mathbb{R}^M$, and $\mu \in \mathbb{R}^M$ and $\Sigma \in \mathbb{R}^{M \times M}$ are given in Eq. (53), calculated by replacing each parametrized unwinded nonlinearity $\varphi(-; \Theta)$ with parameter-less nonlinearity $\varphi(-; \mathring{\Theta})$.

F Programs with Variable Dimensions

Notation In this section, we let $\dim(x)$ denote the dimension of an H-var x.

Before this section, we have mostly assumed that all dimensions in a NETSOR (or NETSOR$^+$) program are equal. This is not necessary, and was done only to more quickly present the main ideas of this work. In general, we can allow the H-vars in a program to vary in dimension, subject to the obvious dimensionality constraints imposed by the different rules:

$$\begin{cases} \text{If } y := \sum_{i=1}^k a_i x^i \text{ or } y := \phi(x^1, \ldots, x^k)\text{, then the } \dim(y) = \dim(x^i) \text{ for each } i. \\ \text{If } y := Wx \text{ and } y' := Wx'\text{, then } \dim(x) = \dim(x') \text{ and } \dim(y) = \dim(y'). \end{cases} \tag{54}$$

Definition F.1. Given an equivalence relation $\simeq$ on the input G-vars of a program, we extend this to an equivalence relation on all H-vars of the program by

$$h \equiv h' \iff h \simeq h' \text{ OR } h \text{ and } h' \text{ are constrained to have the same dimension by (54).} \tag{55}$$

We call any such equivalence class a *Common Dimension Class*, or CDC.

Intuitively, the dimensions of H-vars in each CDC are all the same, and this common dimension is allowed to vary between CDCs.

Example F.2. In Program 1, the CDCs are $\{W^1x, b^1, h^1, x^1\}$ and $\{b^2, v, \tilde{h}^2, h^2, x^2\}$. In Program 2, all G-vars are in the same CDC, and given the body of the program, this is the only way to partition the H-vars into CDCs, because the reuse of W across time step ties all H-var dimensions to be equal.

Assumption F.3. *Fix a* NETSOR *program with some equivalence relation on the input G-vars, and thus with induced CDCs over its H-vars. Assume the dimensions in each CDC are the same, but the dimensions of different CDCs can vary. Suppose for each A-var W : $\mathsf{A}(m', m)$, we sample $W_{\alpha\beta} \sim \mathcal{N}(\sigma_W^2/m)$ for some $\sigma_W^2 > 0$. Suppose further for each CDC $\mathcal{C}$ with dimension n, for each $\alpha \in [n]$, we sample, i.i.d., $\{x_\alpha : x \in \mathcal{C}$ and x is input G-var$\} \sim \mathcal{N}(\mu^{\mathcal{C}}, \Sigma^{\mathcal{C}})$ for some mean $\mu^{\mathcal{C}}$ and covariance $\Sigma^{\mathcal{C}}$ over input G-vars in $\mathcal{C}$.*

Then the following result is an easy extension of Theorem 5.4.

Theorem F.4 (NETSOR Master Theorem; Variable Dimensions)**.** *Fix any* NETSOR *program satisfying Assumption F.3 and with all nonlinearities controlled. For any CDC $\mathcal{C}$, if $g^1, \ldots, g^M$ are all of the G-vars (including all input G-vars) in $\mathcal{C}$, then for any controlled $\psi : \mathbb{R}^M \to \mathbb{R}$, as all dimensions in the program tend to infinity (not just the dimension of $\mathcal{C}$)* [13]*,*

$$\frac{1}{n}\sum_{\alpha=1}^n \psi(g_\alpha^1, \ldots, g_\alpha^M) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu^{\mathcal{C}}, \Sigma^{\mathcal{C}})} \psi(Z) = \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu^{\mathcal{C}}, \Sigma^{\mathcal{C}})} \psi(Z^{g^1}, \ldots, Z^{g^M}),$$

where $\xrightarrow{\text{a.s.}}$ means almost sure convergence, $Z = (Z^{g^1}, \ldots, Z^{g^M}) \in \mathbb{R}^M$, and $\mu^{\mathcal{C}} = \{\mu^{\mathcal{C}}(g^i)\}_{i=1}^M \in \mathbb{R}^M$ and $\Sigma^{\mathcal{C}} = \{\Sigma^{\mathcal{C}}(g^i, g^j)\}_{i,j=1}^M \in \mathbb{R}^{M\times M}$ are given in Eq. (56). See Fig. 1 for an illustration.

Definition F.5. For any CDC $\mathcal{C}$ and G-vars g, g' in $\mathcal{C}$, define recursively

$$\mu^{\mathcal{C}}(g) = \begin{cases} \mu^{\mathcal{C}}(g) & \text{if } g \text{ is input} \\ \sum_i a_i \mu^{\mathcal{C}}(y^i) & \text{if } g = \sum_i a_i y^i\text{, introduced by } \texttt{LinComb}\text{ ,} \\ 0 & \text{otherwise} \end{cases}$$

$$\Sigma^{\mathcal{C}}(g, g') = \begin{cases} \Sigma^{\mathcal{C}}(g, g') & \text{if } g, g' \text{ are inputs} \\ \sum_i a_i \Sigma^{\mathcal{C}}(y^i, g') & \text{if } g = \sum_i a_i y^i\text{, introduced by } \texttt{LinComb} \\ \sum_i a_i \Sigma^{\mathcal{C}}(g, y^i) & \text{if } g' = \sum_i a_i y^i\text{, introduced by } \texttt{LinComb} \\ \sigma_W^2 \mathbb{E}_Z \varphi^h(Z)\varphi^{h'}(Z) & \text{if } g = Wh, g' = Wh'\text{, introduced by } \texttt{MatMul} \text{ w/ same A-var } W \\ 0 & \text{otherwise} \end{cases} \tag{56}$$

where $Z \sim \mathcal{N}(\mu^{\mathcal{C}'}, \Sigma^{\mathcal{C}'})$ with $\mathcal{C}'$ denoting the CDC of h and h'.

Essentially the same proof of Theorem 5.4 goes through for Theorem F.4, by noting that this proof only requires the minimum of all dimensions to go to infinity.

[13]Note that we do not require the dimensions of different CDCs to have a convergent, finite but nonzero, ratio

G Theoretical Tools

In this section, we list a series of theoretical tools needed to prove the Master Theorems.

G.1 Probability Facts

Notations Given two random variables X, Y, and a σ-algebra $\mathcal{A}$, the notation $X \overset{\mathrm{d}}{=}_{\mathcal{A}} Y$ means that for any integrable function ϕ and for any random varible Z measurable on $\mathcal{A}$, $\mathbb{E}\,\phi(X)Z = \mathbb{E}\,\phi(Y)Z$. We say that X is distributed as (or is equal in distribution to) Y conditional on $\mathcal{A}$. In case $\mathcal{A}$ is the trivial σ-algebra, we just write $X \overset{\mathrm{d}}{=} Y$. The expression $X \xrightarrow{\mathrm{d}} Y$ (resp. $X \xrightarrow{\text{a.s.}} Y$) means X converges to Y in distribution (resp. almost surely).

Lemma G.1. *Let $\{X_n\}_{n\geq 1}$ be a sequence of random variables with zero mean. If for some $p \in \mathbb{N}$ and for all n, $\mathbb{E}\,X_n^{2p} \leq cn^{-1-\rho}$, for some $\rho > 0$, then $X_n \to 0$ almost surely.*

Proof. By Markov's inequality, for any $\epsilon > 0$,

$$\Pr(|X_n| > \epsilon) = \Pr(X_n^{2p} > \epsilon^{2p}) \leq \mathbb{E}\,X_n^{2p}/\epsilon^{2p} \leq cn^{-1-\rho}/\epsilon^{2p}$$
$$\sum_n \Pr(|X_n| > \epsilon) \leq \sum_n cn^{-1-\rho}/\epsilon^{2p} < \infty.$$

By Borel-Cantelli Lemma, almost surely, $|X_n| \leq \epsilon$ for all large n. Then, if we pick a sequence $\{\epsilon_k > 0\}_k$ converging to 0, we have that, almost surely, for each k, $|X_n| \leq \epsilon_k$ for large enough n — i.e. almost surely, $X_n \to 0$. ☐

The following is a standard fact about multivariate Gaussian conditioning

Proposition G.2. *Suppose $\mathbb{R}^{n_1+n_2} \ni x \sim \mathcal{N}(\mu, K)$, where we partition $x = (x_1, x_2) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$, $\mu = (\mu_1, \mu_2) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$, and $K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}$. Then $x_1 \overset{\mathrm{d}}{=}_{x_2} \mathcal{N}(\mu|_{x_2}, K|_{x_2})$ where*

$$\mu|_{x_2} = \mu_1 - K_{12}K_{22}^{+}(x_2 - \mu_2)$$
$$K|_{x_2} = K_{11} - K_{12}K_{22}^{+}K_{21}.$$

Lemma G.3 (Stein's lemma)**.** *For jointly Gaussian random variables Z_1, Z_2 with zero mean, and any function $\phi : \mathbb{R} \to \mathbb{R}$ where $\mathbb{E}\,\phi'(Z_1)$ and $\mathbb{E}\,Z_1\phi(Z_2)$ exists, we have*

$$\mathbb{E}\,Z_1\phi(Z_2) = \mathrm{Cov}(Z_1, Z_2)\,\mathbb{E}\,\phi'(Z_2).$$

Proposition G.4 (Convergence of output vector to Gaussian given convergent 2nd moments)**.** *Consider a sequence (in $t \in \mathbb{N}$) of collections of random vectors $\{x^{ab} \in \mathbb{R}^{n_a}\}_{b=1}^{r_a}$, $a = 1, \ldots, m$, where n_a and x^{ab} can depend on t but m and r_a are fixed. Suppose as $t \to \infty$, $\frac{1}{n_a} x^{ab\top} x^{ab'} \xrightarrow{\mathrm{d}} \Sigma^{\infty}_{ab,ab'}$ for some deterministic PSD matrix $\Sigma^{\infty} = \{\Sigma^{\infty}_{ab,a'b'}\}_{a,b,a',b'}$. If $v^a \sim \mathcal{N}(0, \sigma_a^2 I)$ is sampled independently for each a, and independently from $\{x^{ab}\}_{a,b}$, then*

$$\{v^{a\top} x^{ab}/\sqrt{n_a}\}_{a,b} \xrightarrow{\mathrm{d}} \mathcal{N}(0, \Sigma)$$

where the covariance $\Sigma = \{\Sigma_{ab,a'b'}\}_{a,b,a',b'}$ has

$$\Sigma_{ab,a'b'} = \begin{cases} \sigma_a^2 \Sigma^{\infty}_{ab,ab'} & \textit{if } a = a' \\ 0 & \textit{else.} \end{cases}$$

Proof. WLOG, we assume $\sigma_a = 1$ for all $a = 1, \ldots, m$. Let $f : \mathbb{R}^{\sum_a r_a} \to \mathbb{R}$ be a bounded continuous function. We need to show that

$$\mathbb{E}\,f(\{v^{a\top} x^{ab}\}_{a,b}) \to \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(0,\Sigma)} f(Z)$$

Note that if we define the PSD matrix $\hat{\Sigma}$ by $\hat{\Sigma}_{ab,ab'} = \frac{1}{n_a} x^{ab\top} x^{ab'}$ and $\hat{\Sigma}_{ab,a'b'} = 0$ if $a \neq a'$, then

$$\mathbb{E}\,f(\{v^{a\top} x^{ab}\}_{a,b}) = \mathop{\mathbb{E}}_{\hat{\Sigma}} \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(0,\hat{\Sigma})} f(Z)$$

where the distribution over $\hat{\Sigma}$ is induced by the distribution over $\{x^{ab}\}$. The function $\tilde{f}(\hat{\Sigma}) := \mathbb{E}_{Z\sim\mathcal{N}(0,\hat{\Sigma})} f(Z)$ is bounded because f is bounded. Thus, as $\hat{\Sigma} \xrightarrow{\mathrm{d}} \Sigma^\infty$ by assumption, we have

$$\mathbb{E}\, f(\{v^{a\top} x^{ab}\}_{a,b}) = \mathop{\mathbb{E}}_{\hat{\Sigma}} \tilde{f}(\hat{\Sigma}) \to \tilde{f}(\Sigma^\infty) = \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(0,\Sigma^\infty)} f(Z)$$

as $t \to \infty$. □

G.2 Review of Moore-Penrose Pseudoinverse

We recall Moore-Penrose pseudoinverse and some properties of it.

Definition G.5. For $A \in \mathbb{R}^{n\times m}$, a pseudoinverse of A is defined as a matrix $A^+ \in \mathbb{R}^{m\times n}$ that satisfies all of the following criteria

- $AA^+A = A$
- $A^+AA^+ = A^+$
- $(AA^+)^\top = AA^+$
- $(A^+A)^\top = A^+A$

The following facts are standard

- if A has real entries, then so does A^+.
- The pseudoinverse always exists and is unique.
- When A is invertible, $A^+ = A^{-1}$.
- $(A^\top)^+ = (A^+)^\top$, which we denote as $A^{+\top}$.
- $A^+ = (A^\top A)^+ A^\top = A^\top (AA^\top)^+$.
- AA^+ is the orthogonal projector to the column space of A; $I - A^+A$ is the orthogonal project to the null space of A.
- if A has singular value decomposition $A = U\Lambda V$ where U and V are orthogonal and Λ has the singular values on its diagonal, then $A^+ = V^\top \Lambda^+ U^\top$ where Λ^+ inverts all nonzero entries of Λ.
- For any collection of vectors $\{v_i\}_{i=1}^n$ in a Hilbert space, $w \mapsto \sum_{i,j=1}^n v_i (\Sigma^+)_{ij} \langle v_j, w\rangle$, where $\Sigma_{ij} = \langle v_i, v_j\rangle$, is the projection operator to the linear span of $\{v_i\}_{i=1}^n$.

G.3 Gaussian Conditioning Trick

The Gaussian conditioning trick was introduced by Bolthausen [5] for solving the TAP equation in statistical physics. Later, this idea was used in Bayati and Montanari [3] to study the Approximate Message Passing algorithm in compressed sensing.

We present a slightly more general versions of lemmas from Bayati and Montanari [3] that deal with singular matrices.

Lemma G.6. *Let $z \in \mathbb{R}^n$ be a random vector with i.i.d. $\mathcal{N}(0,\sigma^2)$ entries and let $D \in \mathbb{R}^{m\times n}$ be a linear operator. Then for any constant vector $b \in \mathbb{R}^n$ the distribution of z conditioned on $Dz = b$ satisfies:*

$$z \overset{\mathrm{d}}{=}_{Dz=b} D^+ b + \Pi \tilde{z}$$

where D^+ is the (Moore-Penrose) pseudoinverse, Π is the orthogonal projection onto subspace $\{z : Dz = 0\}$, and $\tilde{z}$ is a random vector of i.i.d. $\mathcal{N}(0,\sigma^2)$.

Proof. When $D = [I_{m\times m} | 0_{m\times n-m}]$, this claim is immediate. By rotational symmetry, this shows that, for any vector space $\mathcal{V}$ and vector v orthogonal to it, conditioning z on $\mathcal{V} + v$ yields a Gaussian centered on v with covariance determined by $\Pi_{\mathcal{V}} z$. Then the lemma in the general case is implied by noting that $\{z : Dz = b\}$ can be decomposed as $\{z : Dz = 0\} + D^+ b$. □

Lemma G.7. *Let $A \in \mathbb{R}^{n\times m}$ be a matrix with random Gaussian entries, $A_{ij} \sim \mathcal{N}(0, \sigma^2)$. Consider fixed matrices $Q \in \mathbb{R}^{m\times q}, Y \in \mathbb{R}^{n\times q}, P \in \mathbb{R}^{n\times p}, X \in \mathbb{R}^{m\times p}$. Suppose there exists a solution in A to the equations $Y = AQ$ and $X = A^\top P$. Then the distribution of A conditioned on $Y = AQ$ and $X = A^\top P$ is*

$$A \overset{\mathrm{d}}{=}_{Y=AQ, X=A^\top P} E + \Pi_P^\perp \tilde{A} \Pi_Q^\perp$$

where

$$E = YQ^+ + P^{+\top} X^\top - P^{+\top} P^\top Y Q^+,$$

$\tilde{A}$ is an iid copy of A, and $\Pi_P^\perp = I - \Pi_P$ and $\Pi_Q^\perp = I - \Pi_Q$ in which $\Pi_P = PP^+$ and $\Pi_Q = QQ^+$ are the orthogonal projection to the space spanned by the column spaces of P and Q respectively.

Proof. We apply Lemma G.6 to $D : A \mapsto (AQ, P^\top A)$. The pseudoinverse of D applied to $(Y, X^\top)$ can be formulated as the unique solution of

$$\operatorname*{argmin}_A \left\{ \|A\|_F^2 : AQ = Y, P^\top A = X^\top \right\}$$

where $\| - \|_F$ denotes Frobenius norm. We check that E is a 1) a solution to $AQ = Y, P^\top A = X^\top$ and 2) the minimal norm solution.

We have $EQ = YQ^+Q + P^{+\top} X^\top Q - P^{+\top} P^\top Y Q^+ Q$. Note that $YQ^+Q = Y$ because $Y = AQ \implies YQ^+Q = AQQ^+Q = AQ = Y$. So $EQ = Y + P^{+T}(X^\top Q - P^\top Y)$. But $X^\top Q = P^\top AQ = P^\top Y$, so $EQ = Y$ as desired. A similar, but easier reasoning, gives $P^\top E = X^\top$. This verifies that E is a solution.

To check that E is minimal norm, we show that it satisfies the stationarity of the Lagrangian

$$L(A, \Theta, \Gamma) = \|A\|_F^2 + \langle \Theta, Y - AQ \rangle + \langle \Gamma, X - A^\top P \rangle.$$

So $\frac{\partial L}{\partial A} = 0 \implies 2A = \Theta Q^\top + P \Gamma^\top$ for some choices of $\Theta \in \mathbb{R}^{n\times q}$ and $\Gamma \in \mathbb{R}^{m\times p}$. For $\Theta = 2Y(Q^\top Q)^+$ and $\Gamma^\top = 2(P^\top P)^+[X^\top - P^\top Y Q^\top]$, we can check that

$$\begin{aligned}\Theta Q^\top + P\Gamma^\top &= 2Y(Q^\top Q)^+ Q^\top + 2P(P^\top P)^+[X^\top - P^\top Y Q^+] \\ &= 2YQ^+ + 2P^{+\top} X^\top - 2P^{+\top} P^\top Y Q^+ \\ &= 2E\end{aligned}$$

as desired. ☐

G.4 α-Controlled Functions

We generalize Definition 5.3 slightly as follows.

Definition G.8 (α-controlled). For $\alpha > 0$, a function $\phi : \mathbb{R}^k \to \mathbb{R}$ is said to be *α-controlled* if for some $C, c > 0$, $|\phi(x)| \le e^{C \sum_{i=1}^k |x_i|^\alpha + c}$ for all $x \in \mathbb{R}^k$.

We present a few helper lemmas to facilitate our reasoning with α-controlled functions. The next lemma is easy to show using the equivalence of norms in finite dimensional Euclidean space.

Lemma G.9. *Let $\phi : \mathbb{R}^k \to \mathbb{R}$. The following are equivalent*

1. *ϕ is α-controlled*
2. *For some $p \ge 1$ and some $g(x) = o_{\|x\|_p \to \infty}(\|x\|_p^\alpha)$, $C, c > 0$, $|\phi(x)| \le e^{C\|x\|_p^\alpha + g(x)}$*
3. *For all $p \ge 1$, there is some $C, c > 0$, $|\phi(x)| \le e^{C\|x\|_p^\alpha + c}$*

Lemma G.10. *Let $\mathsf{C}_\alpha^k : \mathbb{R}^{\ge 0} \to \mathbb{R}, c \mapsto \mathbb{E}_{z\sim\mathcal{N}(0, I_k)} e^{c\|z\|_2^\alpha}$. Then*

1. *$\mathsf{C}_\alpha^k < \infty$ iff $\alpha < 2$*
2. *for $\alpha \ge 1$,*

$$\mathop{\mathbb{E}}_{z\sim\mathcal{N}(\mu,\Sigma)} e^{C\|z\|_2^\alpha} \le e^{C\|\mu\|_2^\alpha} \mathsf{C}_\alpha^k (C\alpha \|\Sigma\|_2^{\alpha/2})$$

where $\|\Sigma\|_2$ denotes the spectral norm of Σ.

3. *for any α-controlled $\phi : \mathbb{R}^k \to \mathbb{R}$ with $\alpha \geq 1$, there is $C > 0$ such that for all $\mu \in \mathbb{R}^k$ and $k \times k$ PSD matrix Σ,*

$$\mathop{\mathbb{E}}_{z \sim \mathcal{N}(\mu,\Sigma)} |\phi(z)| \leq C e^{C\|\mu\|_2^\alpha} \mathsf{C}_\alpha^k(C\alpha\|\Sigma\|_2^{\alpha/2})$$

where $\|\Sigma\|_2$ denotes the spectral norm of Σ.

Note that the RHS is a montonic function in $\|\mu\|_2$ and $\|\Sigma\|_2$, in the sense that if $\|\mu\|_2$ and $\|\Sigma\|_2$ don't decrease, then the RHS will not decrease either.

Proof. The first claim is obvious and the third follows from the second easily. For the second,

$$\begin{aligned}
\mathop{\mathbb{E}}_{z \sim \mathcal{N}(\mu,\Sigma)} e^{C\|z\|_2^\alpha} &\leq \mathop{\mathbb{E}}_{z \sim \mathcal{N}(0,I)} e^{C\|\sqrt{\Sigma}z+\mu\|_2^\alpha} \\
&\leq \mathop{\mathbb{E}}_{z \sim \mathcal{N}(0,I)} e^{C\alpha\left(\|\sqrt{\Sigma}z\|_2^\alpha+\|\mu\|_2^\alpha\right)} \\
&\leq e^{C\|\mu\|_2^\alpha} \mathop{\mathbb{E}}_{z \sim \mathcal{N}(0,I)} e^{C\alpha\|\Sigma\|_2^{\alpha/2}\|z\|_2^\alpha} \\
&= e^{C\|\mu\|_2^\alpha} \mathsf{C}_\alpha^k(C\alpha\|\Sigma\|_2^{\alpha/2}).
\end{aligned}$$

□

H Proof of NETSOR Master Theorem

In this section, we prove Theorem E.3, i.e. the Master Theorem for programs without `LinComb`. By Remark E.2, this would also show Theorem 5.4.

A Bit of Notation and Terminology Note that, for each n, the randomness of our program specified by Theorem 5.4 comes from the sampling of the input variables. Let $\mathcal{U}$ be the product space obtained from multiplying together the corresponding probability space for each n. Each sample from this product probability space thus correspond to a sequence $\{S(n)\}_n$ of instantiatiations of input variables. Below, when we say "almost surely" (often abbreviated "a.s."), we mean "almost surely over the probability of $\mathcal{U}$." We will also often make statements of the form

almost surely (or, a.s.), for all large n, $\quad \mathcal{A}(n)$ is true

where $\mathcal{A}(n)$ is a claim parametrized by n. This means that for all but a $\mathcal{U}$-probability-zero set of sequences $\{S(n)\}_n$ of input variable instantiations, $\mathcal{A}(n)$ is true for large enough n. Note that the order of the qualifiers is very important here.

We induct, but on what? A natural way of going about proving Theorem 5.4 is by inducting on the number of variables in a program. It turns out this is not enough to prove our claim in its full generality (see below), and it would be more fruitful to perform a simultaneous induction on our claim (Moments) along with another statement, parametrized by m,

Moments (m) For any controlled $\psi : \mathbb{R}^m \to \mathbb{R}$, as $n \to \infty$,

$$\frac{1}{n}\sum_{\alpha=1}^n \psi(g_\alpha^1, \ldots, g_\alpha^m) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu,\Sigma)} \psi(Z).$$

CoreSet (m) There exists a "core set" $\mathcal{M} \subseteq [m]$ such that,

Basis (m) almost surely, for large enough n, for every $i \in [m]$, there exist *unique* constants (not depending on n) $\{a_j\}_{j\in\mathcal{M}}$ such that $g^i = \sum_{j\in\mathcal{M}} a_j g^j$. Note the uniqueness implies that $\{g^i\}_{i\in\mathcal{M}}$ is linearly independent.

NullAvoid (m) for every triangular array of Lesbegue measure zero sets $\{A_{n\alpha} \in \mathbb{R}^{\mathcal{M}}\}_{n\in\mathbb{N},\alpha\in[n]}$, almost surely for all large enough n, for all $\alpha \in [n]$, we have

$$\{g_\alpha^i\}_{i\in\mathcal{M}} \notin A_{n\alpha}.$$

In other words, the values $\{g_\alpha^i\}_{\alpha\in\mathcal{M}}$ of the core set "avoid" Lebesgue measure zero sets asymptotically. Intuitively, this says that the distribution of these values are not singular. (Note the LHS depends on n although we are suppressing it notationally)

Let us explain in brief why we need to consider CoreSet satisfying Basis and NullAvoid.

- Basis reduces the consideration of Moments to only the core set G-vars, since every other G-var is asymptotically a linear combination of them.
- When we apply the Gaussian conditioning technique Proposition G.2, we need to reason about the pseudo-inverse Λ^+ of some submatrix Λ of a covariance matrix. Each entry of Λ is of the form $\frac{1}{n}\sum_{\alpha=1}^{n}\phi_i(g^1_\alpha,\ldots,g^{m-1}_\alpha)\phi_j(g^1_\alpha,\ldots,g^{m-1}_\alpha)$ for a collection of controlled scalar functions $\{\phi_i\}_i$. This Λ will be a random variable which converges a.s. to a determinstic limit $\mathring{\Lambda}$ as $n\to\infty$. It should be generically true that $\Lambda^+ \xrightarrow{\text{a.s.}} \mathring{\Lambda}^+$ as well, which is essential to make the Gaussian conditioning argument go through. But in general, this is guaranteed only if Λ's rank doesn't drop suddenly in the $n\to\infty$ limit. We thus need to guard against the possibility that $g^1,\ldots,g^m$, in the limit, suddenly concentrate on a small set on which $\{\phi_i(g^1,\ldots,g^m)\}_i$ are linearly dependent. This is where NullAvoid comes in. It tells us that $g^1,\ldots,g^m$ will avoid any such small set asymptotically, so that indeed the rank of Λ will not drop in the limit.

Proof organization We will show that Moments and CoreSet are true for input variables, as the base case, and

$$\text{Moments}(m-1) \text{ and } \text{CoreSet}(m-1) \implies \text{Moments}(m) \text{ and } \text{CoreSet}(m)$$

as the inductive step. By induction, we obtain Moments(M), which is Theorem 5.4.

The base cases are easy and we will dispatch with them immediately after this in Appendix H.1, but the inductive step is much more complicated, and we will need to set up notation in Appendix H.2. During this setup, we prove some basic limit theorems using the induction hypothesis. However, the full generality of these claims requires some consequences of CoreSet, which we call "rank stability" and "zero stability" (related to Assumption C.3). These notions are introduced and proved in Appendix H.3.

We would then finally be able to handle the inductive steps at this point. We first prove

$$\text{Moments}(m-1) \text{ and } \text{CoreSet}(m-1) \implies \text{CoreSet}(m)$$

in Appendix H.4 because it is easier. Then we prove

$$\text{Moments}(m-1) \text{ and } \text{CoreSet}(m-1) \implies \text{Moments}(m)$$

in Appendix H.5.

H.1 Base Cases: Moments and CoreSet for Input Variables

Base case: Moments(input vars) Suppose the input variables are $x^1,\ldots,x^k : \mathsf{G}(n)$ (so that $\mu^{\text{in}}\in\mathbb{R}^k, \Sigma^{\text{in}}\in\mathbb{R}^{k\times k}$). We need to show that for any controlled function $\psi:\mathbb{R}^k\to\mathbb{R}$,

$$\frac{1}{n}\sum_{\alpha=1}^{n}\psi(x^1_\alpha,\ldots,x^k_\alpha) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)}\psi(Z),$$

where ψ on the RHS ignores all coordinates corresponding to non-input G-vars. Since μ and Σ restricted to input variables are just μ^{in} and Σ^{in} (see Eq. (2)), the RHS expectation is just

$$\mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)}\psi(Z) = \mathop{\mathbb{E}}_{Z^{\text{in}}\sim\mathcal{N}(\mu^{\text{in}},\Sigma^{\text{in}})}\psi(Z^{\text{in}})$$

and the almost sure convergence we desire is just a result of the law of large numbers.

Base Case: CoreSet(input vars) Let $x^1,\ldots,x^k$ be the input G-vars as above. Pick the core set $\mathcal{M}$ to be any subset of $[k]$ such that $\operatorname{rank}\Sigma^{\text{in}}|_{\mathcal{M}} = \operatorname{rank}\Sigma^{\text{in}}$. Then it's straightforward to verify Basis and NullAvoid.

H.2 Inductive Case: Setup

We now assume Moments$(m-1)$ and CoreSet$(m-1)$ and want to reason about g^m to show Moments(m) and CoreSet(m). Suppose

$$g^m := Ah \quad \text{where} \quad A : \mathsf{A}(n,n) \text{ and } h : \mathsf{H}(n) \text{ was introduced by } h := \phi(g^1, \ldots, g^{m-1})$$

(WLOG padding coordinates if necessary; if $h = g^i$ is a G-var, then pretend ϕ just projects to the ith coordinate). For brevity, we will just write $g = g^m$. Consider all previous instances where A is used:

$$\hat{g}^i := A\hat{h}^i, i = 1, \ldots, r.$$

Define

$$\hat{G} \stackrel{\text{def}}{=} [\hat{g}^1|\ldots|\hat{g}^r] \in \mathbb{R}^{n\times r}, \hat{H} \stackrel{\text{def}}{=} [\hat{h}^1|\ldots|\hat{h}^r]. \tag{57}$$

We will also use $\hat{G}$ to denote the *set* of G-vars $\{\hat{g}^1, \ldots, \hat{g}^r\}$ when we later write expressions like $\Sigma(\hat{G}, \hat{G})$. Let $\mathcal{B}$ be the σ-algebra spanned by all previous G-vars $g^1, \ldots, g^{m-1}$ (and hence also all previous H-vars). Conditioning on $\mathcal{B}$, A is constrained by $\hat{G} = A\hat{H}$, and we have by Lemma G.7,

$$g \stackrel{\text{d}}{=}_{\mathcal{B}} (\hat{G}\hat{H}^+ + \tilde{A}\Pi^\perp_{\hat{H}})h$$

where $\tilde{A}$ is an independent copy of A and $\Pi_{\hat{H}} = \hat{H}\hat{H}^+ = \hat{H}(\hat{H}^\top\hat{H})^+\hat{H}^\top$ is the projection to the column space of $\hat{H}$.

If we define

$$\omega \stackrel{\text{def}}{=} \hat{G}\hat{H}^+h, \quad \sigma \stackrel{\text{def}}{=} \sigma_A\sqrt{\|\Pi^\perp_{\hat{H}}h\|^2/n} \tag{58}$$

then

$$g \stackrel{\text{d}}{=}_{\mathcal{B}} \omega + \sigma y, \text{ with } y \sim \mathcal{N}(0, I_n) \tag{59}$$

For brevity, we will define the following matrices and vectors of fixed dimension

$$\hat{\Lambda} \stackrel{\text{def}}{=} \hat{H}^\top\hat{H}/n \in \mathbb{R}^{r\times r}, \quad \hat{\eta} \stackrel{\text{def}}{=} \hat{H}^\top h/n \in \mathbb{R}^r. \tag{60}$$

Suppose $\hat{h}^i$ was introduced by $\hat{h}^i := \hat{\phi}^i(g^1, \ldots, g^M)$, where $\hat{\phi}^i$ depends at most on $g^1, \ldots, g^{m-1}$. By induction hypothesis Moments$(m-1)$, $\hat{\Lambda}$ and $\hat{\eta}$ all converge a.s. to corresponding limit values $\mathring{\hat{\Lambda}}$ and $\mathring{\hat{\eta}}$, since their entries are moments of $Z^1, \ldots, Z^{m-1}$:

$$\hat{\Lambda}_{ij} \xrightarrow{\text{a.s.}} \mathring{\hat{\Lambda}}_{ij} \stackrel{\text{def}}{=} \mathbb{E}\,\hat{\phi}^i(Z)\hat{\phi}^j(Z) = (\sigma_A)^{-2}\Sigma(\hat{g}^i, \hat{g}^j)$$
$$\hat{\eta}_i \xrightarrow{\text{a.s.}} \mathring{\hat{\eta}}_i \stackrel{\text{def}}{=} \mathbb{E}\,\hat{\phi}^i(Z)\phi(Z) = (\sigma_A)^{-2}\Sigma(\hat{g}^i, g).$$

It turns out that, as a consequence of Lemma H.4 below, a.s. for all large enough n, $\operatorname{rank}\hat{\Lambda} = \operatorname{rank}\mathring{\hat{\Lambda}}$. Therefore, as pseudoinverse is continuous on matrices of fixed rank, we get the following proposition

Proposition H.1. $\hat{\Lambda}^+ \xrightarrow{\text{a.s.}} \mathring{\hat{\Lambda}}^+$.

Using this proposition, we compute the limits of the conditional mean ω and variance σ^2.

Lemma H.2. $\sigma^2 \xrightarrow{\text{a.s.}} \mathring{\sigma}^2 \stackrel{\text{def}}{=} \Sigma(g, g) - \Sigma(g, \hat{G})\Sigma(\hat{G}, \hat{G})^+\Sigma(\hat{G}, g)$

Proof. Note that

$$\sigma^2 = \frac{\sigma_A^2}{n}(h^\top h - h^\top\Pi_{\hat{H}}h) = \frac{\sigma_A^2}{n}(h^\top h - h^\top\hat{H}(\hat{H}^\top\hat{H})^+\hat{H}^\top h) = \frac{\sigma_A^2}{n}(h^\top h - \hat{\eta}^\top\hat{\Lambda}^+\hat{\eta}).$$

Because ϕ is polynomially-bounded, so is $\phi(z)^2$ as well. By induction hypothesis,

$$\frac{1}{n}h^\top h = \frac{1}{n}\sum_{\alpha=1}^n \phi(g^1_\alpha, \ldots, g^{m-1}_\alpha)^2 \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)} \phi(Z)^2 = \sigma_A^{-2}\Sigma(g, g).$$

Likewise, $\hat{\eta} \xrightarrow{\text{a.s.}} \mathring{\hat{\eta}}$ and $\hat{\Lambda} \xrightarrow{\text{a.s.}} \mathring{\hat{\Lambda}}$. By Proposition H.1, $\hat{\Lambda}^+ \xrightarrow{\text{a.s.}} \mathring{\hat{\Lambda}}^+$. Combining all of these limits together yields the desired claim. □

Lemma H.3. *Let $v \overset{\text{def}}{=} \hat{\Lambda}^+ \hat{\eta}$, so that $v \xrightarrow{\text{a.s.}} \mathring{v} \overset{\text{def}}{=} \mathring{\hat{\Lambda}}^+ \mathring{\hat{\eta}}$. Then for some vector $\hat{\varepsilon} \in \mathbb{R}^r$ that go to 0 a.s. with n, $\omega = Eh = \hat{G}(\mathring{v} + \hat{\varepsilon})$*

Proof. Using Eq. (60), we can re-express ω as $\omega = \hat{G}\hat{\Lambda}^+\hat{\eta}$. By Proposition H.1, $\hat{\Lambda}^+ \xrightarrow{\text{a.s.}} \mathring{\hat{\Lambda}}^+$, so that setting $\hat{\varepsilon} \overset{\text{def}}{=} v - \mathring{v}$, we get $\hat{\varepsilon} \xrightarrow{\text{a.s.}} 0$. Thus, $\omega = \hat{G}(\mathring{v} + \hat{\varepsilon})$ as desired. □

H.3 Rank Stability and Zero Stability

In this section, we prove the following consequence of CoreSet$(m-1)$ and Moments$(m-1)$.

Lemma H.4 (Rank Stability)**.** *For any collection of controlled functions $\{\psi_j : \mathbb{R}^{m-1} \to \mathbb{R}\}_{j=1}^l$, let $K \in \mathbb{R}^{l \times l}$ be the random matrix (depending on n) defined by*

$$K_{ij} = \frac{1}{n} \sum_{\alpha=1}^{n} \psi_i(g_\alpha^1, \dots, g_\alpha^{m-1}) \psi_j(g_\alpha^1, \dots, g_\alpha^{m-1}).$$

By Moments$(m-1)$,

$$K \xrightarrow{\text{a.s.}} \mathring{K}$$

for some matrix $\mathring{K} \in \mathbb{R}^{l \times l}$.

1. *Then, almost surely, for large enough n,*

$$\ker K = \ker \mathring{K}, \quad \operatorname{im} K = \operatorname{im} \mathring{K}, \quad \textit{and} \quad \operatorname{rank} K = \operatorname{rank} \mathring{K}.$$

   Here ker *denotes null space and* im *denotes image space.*

2. *Suppose $I \subseteq [l]$ is any subset such that $\mathring{K}|_I$, the restriction of $\mathring{K}$ to rows and columns corresponding to I, satisfies*

$$|I| = \operatorname{rank} \mathring{K}|_I = \operatorname{rank} \mathring{K}.$$

   There are unique coefficients $\{F_{ij}\}_{i \in [l], j \in I}$ that expresses each row of $\mathring{K}$ as linear combinations of rows corresponding to I:

$$\forall i \in [l], \quad \mathring{K}_i = \sum_{j \in I} F_{ij} \mathring{K}_j.$$

   Then, a.s. for all large n, for all $\alpha \in [n]$,

$$\psi_i(g_\alpha^1, \dots, g_\alpha^{m-1}) = \sum_{j \in I} F_{ij} \psi_j(g_\alpha^1, \dots, g_\alpha^{m-1}).$$

This will be primarily a corollary of the following Lemma H.5.

Lemma H.5 (Zero Stability)**.** *If $\psi : \mathbb{R}^{m-1} \to \mathbb{R}^{\geq 0}$ is a nonnegative function such that*

$$\frac{1}{n} \sum_{\alpha=1}^{n} \psi(g_\alpha^1, \dots, g_\alpha^{m-1}) \xrightarrow{\text{a.s.}} 0$$

then, almost surely, for large enough n,

$$\psi(g_\alpha^1, \dots, g_\alpha^{m-1}) = 0$$

for all $\alpha \in [n]$.

We give the proof of Lemma H.4 now, assuming Lemma H.5.

Proof. Let $v \in \mathbb{R}^l$ be in the null space of $\mathring{K}$, i.e. $v^\top \mathring{K} v = 0$. Then we also have $v^\top K v \xrightarrow{\text{a.s.}} v^\top \mathring{K} v = 0$. But

$$v^\top K v = \frac{1}{n} \sum_{\alpha=1}^{n} \Psi(g_\alpha^1, \dots, g_\alpha^{m-1}), \quad \text{where} \quad \Psi(g_\alpha^1, \dots, g_\alpha^{m-1}) \overset{\text{def}}{=} \left(\sum_{i=1}^{} v_i \psi_i(g_\alpha^1, \dots, g_\alpha^{m-1}) \right)^2$$

and Ψ is a nonnegative function. By Lemma H.5, we have that: almost surely, for large enough n,

$$\Psi(g^1_\alpha, \ldots, g^{m-1}_\alpha) = 0 \quad \text{for all } \alpha \in [n] \qquad \Longrightarrow \quad v^\top K v = 0$$

Claim 1. If we apply this argument to a basis $\{v^1, \ldots, v^t\}$ of $\ker \mathring{K}$, then we get,

$$\text{a.s. for all large } n, \quad \ker \mathring{K} \subseteq \ker K,$$

so that

$$\text{a.s. for all large } n, \quad \operatorname{rank} \mathring{K} \geq \operatorname{rank} K.$$

Because the rank function is lower semicontinuous (i.e. the rank can drop suddenly, but cannot increase suddenly), and $K \xrightarrow{\text{a.s.}} \mathring{K}$, we also have

$$\text{a.s. for all large } n, \quad \operatorname{rank} \mathring{K} \leq \operatorname{rank} K.$$

Combined with the above, this gives the desired result on rank. The equality of null space then follows from the equality of rank, and the equality of image space follows immediately, as the image space is the orthogonal complement of the null space.

Claim 2. If we apply the above argument to each v^i defined by inner product as

$$\forall x \in \mathbb{R}^l, \quad x^\top v^i \overset{\text{def}}{=} x_i - \sum_{j \in I} F_{ij} x_j,$$

(note that only for $i \notin I$ is v^i nonzero), then we have, a.s. for large n, $v^{i\top} K v^i = 0$, or

$$\psi_i(g^1_\alpha, \ldots, g^{m-1}_\alpha) = \sum_{j \in I} F_{ij} \psi_j(g^1_\alpha, \ldots, g^{m-1}_\alpha).$$

□

In the rest of this section, we prove Lemma H.5. It helps to first show that the linear relations given in Basis carries over to the $n \to \infty$ limit.

Proposition H.6. *Let $\Sigma|_\mathcal{M}$ be the submatrix of Σ with rows and columns corresponding to $\{g^i : i \in \mathcal{M}\}$. Then $\operatorname{rank} \Sigma = \operatorname{rank} \Sigma|_\mathcal{M} = |\mathcal{M}|$. Furthermore, if $Z = (Z^1, \ldots, Z^{m-1}) \sim \mathcal{N}(\mu|_{m-1}, \Sigma|_{m-1})$, where $\mu|_{m-1}, \Sigma|_{m-1}$ are the restrictions of μ, Σ to $g^1, \ldots, g^{m-1}$, then*

$$Z^i \overset{\text{d}}{=} \sum_{j \in \mathcal{M}} a_j Z^j$$

where $\{a_j\}_{j \in \mathcal{M}}$ are the coefficients corresponding to g^i given in Basis.

Proof. By Basis property, each $g^i, i \in \mathcal{M}$, has a set of unique constants $\{a_j\}_{j \in \mathcal{M}}$ (independent of n) such that, almost surely, for large enough n,

$$g^i = \sum_{j \in \mathcal{M}} a_j g^j.$$

Let $\psi(x^1, \ldots, x^{m-1}) \overset{\text{def}}{=} (x^i - \sum_{j \in \mathcal{M}} a_j x^j)^2$. Then by Basis$(m-1)$ and Moments$(m-1)$,

$$\frac{1}{n} \sum_{\alpha=1}^{n} \psi(g^1_\alpha, \ldots, g^{m-1}_\alpha) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu|_{m-1}, \Sigma|_{m-1})} \psi(Z) = 0.$$

where $\mu|_{m-1}, \Sigma|_{m-1}$ are the restrictions of μ, Σ to $g^1, \ldots, g^{m-1}$. This implies that for $Z = (Z^1, \ldots, Z^{m-1}) \sim \mathcal{N}(\mu|_{m-1}, \Sigma|_{m-1})$,

$$Z^i \overset{\text{d}}{=} \sum_{j \in \mathcal{M}} a_j Z^j.$$

Repeating this argument for all $i \in [m-1]$ implies that $\{Z^j\}_{j \in \mathcal{M}}$ is a "spanning set" of $Z^1, \ldots, Z^{m-1}$. Furthermore, by the uniqueness of the coefficients, we also have that $\{Z^j\}_{j \in \mathcal{M}}$ is linearly independent as well. This then implies the rank consequence we want. □

Now we show Lemma H.5.

Proof of Lemma H.5. By Moments$(m-1)$,

$$\frac{1}{n}\sum_{\alpha=1}^{n}\psi(g_\alpha^1,\ldots,g_\alpha^{m-1}) \to \mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu|_{m-1},\Sigma|_{m-1})}\psi(Z).$$

By Proposition H.6, if $Z\sim\mathcal{N}(\mu|_{m-1},\Sigma|_{m-1})$ and $Z|_{\mathcal{M}}$ is the part of Z corresponding to $\mathcal{M}$, then

$Z|_{\mathcal{M}}$ **has density.** The law of $Z|_{\mathcal{M}}$ (namely $\mathcal{N}(\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}})$, where $\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}}$ are the restriction of μ and Σ to $\mathcal{M}$) is absolutely continuous against the Lebesgue measure of $\mathbb{R}^{\mathcal{M}}$ and vice versa, so that a set of Lebesgue measure zero is measure zero under $\mathcal{N}(\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}})$, and vice versa; and

$Z|_{\mathcal{M}}$ **is basis of** Z**.** Basis yields a linear function λ such that $\lambda(\{g_\alpha^j\}_{j\in\mathcal{M}})=\{g_\alpha^i\}_{i=1}^{m-1}$ for all α, almost surely asymptotically, and $\lambda(Z|_{\mathcal{M}})\overset{\mathrm{d}}{=}Z$, so that

$$\mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu|_{m-1},\Sigma|_{m-1})}\psi(Z)=\mathop{\mathbb{E}}_{Z'\sim\mathcal{N}(\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}})}\psi\circ\lambda(Z').$$

This expectation is 0 by our premise.

Because ψ, and thus $\psi\circ\lambda$, is a nonnegative function, the nullity of the expectation implies that, other than a set U of $\mathcal{N}(\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}})$-measure zero, $\psi\circ\lambda$ is 0. This set U also has Lebesgue measure zero as $Z|_{\mathcal{M}}$ has density, by our reasoning above.

If in NullAvoid, we set $A_{n\alpha}=U$ for all n and all $\alpha\in[n]$, then we get that: almost surely, for all large enough n, for all $\alpha\in[n]$,

$$\{g_\alpha^i\}_{i\in\mathcal{M}}\notin U \iff \psi\circ\lambda(\{g_\alpha^i\}_{i\in\mathcal{M}})=0 \iff \psi(g_\alpha^1,\ldots,g_\alpha^{m-1})=0,$$

as desired. ☐

H.4 Inductive Step: CoreSet(m)

In this section, we show

$$\text{Moments}(m-1)\text{ and CoreSet}(m-1)\implies\text{CoreSet}(m).$$

More explicitly, we need to think about whether to add m to the core set $\mathcal{M}$ of $[m-1]$ in order to maintain the Basis and NullAvoid properties.

We proceed by casework on whether $\mathring{\sigma}=0$.

H.4.1 If $\mathring{\sigma}=0$

We will show that the core set properties are maintained if we don't add m to the core set.

Consider the space $\mathcal{L}\overset{\text{def}}{=}L^2(\mathcal{N}(\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}}))$ of square-integrable real functions against the measure $\mathcal{N}(\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}})$ defined on $\mathbb{R}^{\mathcal{M}}$. Let $\langle\phi,\psi\rangle=\mathbb{E}_{Y\sim\mathcal{N}(\mu|_{\mathcal{M}},\Sigma|_{\mathcal{M}})}\,\phi(Y)\psi(Y)$ be the inner product of this space. Just like in a finite-dimensional inner product space, given a finite collection of functions $S=\{\psi^i\}_{i=1}^k$, the orthogonal projection operator Π_S to the span of S (inside $\mathcal{L}$) is given by

$$\Pi_S\phi=\sum_{i=1}^{k}a_i\psi^i,$$

for any $\phi\in\mathcal{L}$, where

$$a=\Lambda^+b\in\mathbb{R}^k,$$
$$b_j=\langle\psi^j,\phi\rangle, b\in\mathbb{R}^k,$$
$$\Lambda_{ij}=\langle\psi^i,\psi^j\rangle,\Lambda\in\mathbb{R}^{k\times k}.$$

Recall that $g = Ah$ where h was introduced by $h := \phi(g^1, \ldots, g^{m-1})$, for some controlled ϕ, and likewise $\hat{g}^i = A\hat{h}^i$ where $\hat{h}^i = \hat{\phi}^i(g^1, \ldots, g^{m-1})$, for each $i \in [r]$. By Basis, we know that, a.s. for large enough n, each of $g^1, \ldots, g^{m-1}$ is a (unique, constant-in-n) linear combination of $\{g^j\}_{j\in\mathcal{M}}$. Therefore, we can express

$$h = \underline{\phi}(\{g^j\}_{j\in\mathcal{M}}), \quad \text{and} \quad \forall i \in [r], \hat{h}^i = \underline{\hat{\phi}}^i(\{g^j\}_{j\in\mathcal{M}})$$

for some functions $\underline{\phi}, \underline{\hat{\phi}}^i \in \mathcal{L}$. For convenience, set $S \stackrel{\text{def}}{=} \{\underline{\hat{\phi}}^i\}_i$.

One can see then, as in the proof of Lemma H.2,

$$\mathring{\sigma}^2 = \sigma_A^2(\mathbb{E}\,\phi(Z)^2 - \mathring{\eta}^\top \mathring{\hat{\Lambda}}^+ \mathring{\eta}) = \sigma_A^2(\langle \underline{\phi}, \underline{\phi}\rangle - \langle \underline{\phi}, \Pi_S \underline{\phi}\rangle)$$

by expanding the definition of $\mathring{\eta}$ and $\mathring{\hat{\Lambda}}$. Therefore, $\mathring{\sigma} = 0$ implies that

$$\langle \underline{\phi}, \underline{\phi}\rangle = \langle \underline{\phi}, \Pi_S \underline{\phi}\rangle$$

so that: after changing its values on a set U of measure zero under $\mathcal{N}(\mu|_\mathcal{M}, \Sigma|_\mathcal{M})$ (and thus also under Lebesgue measure by Lemma H.4), $\underline{\phi}$ is a linear combination of $\{\underline{\hat{\phi}}^i\}_{i=1}^r$, i.e.

$$\forall \boldsymbol{x} \notin U, \underline{\phi}(\boldsymbol{x}) = \sum_{i\in[r]} c_i \underline{\hat{\phi}}^i(\boldsymbol{x})$$

for some coefficients $\{c_i\}_{i\in[r]}$. By NullAvoid applied to $A_{n\alpha} = U$ for all n and $\alpha \in [n]$, we also have that: a.s. for large enough n,

$$\phi(g^1, \ldots, g^\alpha) = \underline{\phi}(\{g^j\}_{j\in\mathcal{M}}) = \sum_{i\in[r]} c_i \underline{\hat{\phi}}^i(\{g^j\}_{j\in\mathcal{M}}) = \sum_{i\in[r]} c_i \hat{\phi}^i(g^1, \ldots, g^\alpha),$$

and therefore, under the same condition, (recall A is the matrix giving rise to g in $g := Ah$)

$$g = A\phi(g^1, \ldots, g^\alpha) = \sum_{i\in[r]} c_i A\hat{\phi}^i(g^1, \ldots, g^\alpha) = \sum_{i\in[r]} c_i \hat{g}^i.$$

This shows that, if we keep the core set as $\mathcal{M}$, then Basis is still satisfied. Since the core set is not changing, NullAvoid just follows from the induction hypothesis.

For usage later in the proof of Moments(m), we record our observation here as follows

Lemma H.7. *If $\mathring{\sigma} = 0$, then there are coefficients $\{c_i\}_{i=1}^r$ such that a.s. for large enough n,*

$$g = \sum_{i\in[r]} c_i \hat{g}^i.$$

H.4.2 If $\mathring{\sigma} > 0$

It's clear that g cannot be in the linear span of $\{\hat{g}^i\}_{i\in[r]}$ asymptotically, so we will add g to the core set, and the Basis property follows immediately. In the below, we shall write $\mathcal{M}$ for the old core set, and $\mathcal{M}' \stackrel{\text{def}}{=} \mathcal{M} \cup \{g\}$ for the new one.

It remains to show NullAvoid for $\mathcal{M}'$. Because the conditional variance of g_α^m given $g^1, \ldots, g^{m-1}$ is σ^2, and because $\mathring{\sigma} > 0$, this assumption implies that, a.s. for all large enough n,

$$g_\alpha^m | g^1, \ldots, g^{m-1} \text{ has density for all } \alpha \in [n]. \tag{61}$$

By "has density" here, we in particular mean that any Lesbegue measure zero set in $\mathbb{R}$ has zero probability under the conditional distribution of g_α^m given $g^1, \ldots, g^{m-1}$.

Now, to prove NullAvoid holds for $\mathcal{M}'$: Let $\{A_{n\alpha} \subseteq \mathbb{R}^{\mathcal{M}'}\}_{n\in\mathbb{N},\alpha\in[n]}$ be a triangular array of Lesbegue measure zero sets. For each $A_{n\alpha}$, define $B_{n\alpha} \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathbb{R}^\mathcal{M} : \lambda(A_{n\alpha}|_{\boldsymbol{x}}) \neq 0\}$, where $A_{n\alpha}|_{\boldsymbol{x}} = \{y \in \mathbb{R} : (\boldsymbol{x}, y) \in A_{n\alpha} \subseteq \mathbb{R}^\mathcal{M} \times \mathbb{R}\}$ is the "slice" of $A_{n\alpha}$ at $\boldsymbol{x}$, and λ is the 1-dimensional Lebesgue measure. Because each $A_{n\alpha}$ has measure zero in $\mathbb{R}^{\mathcal{M}'}$, necessarily each $B_{n\alpha}$ also has measure zero in $\mathbb{R}^\mathcal{M}$. Applying NullAvoid to the triangular array $\{B_{n\alpha} \subseteq \mathbb{R}^\mathcal{M}\}_{n\in\mathbb{N},\alpha\in[n]}$, we get that: a.s. for large enough n,

$$\forall \alpha \in [n], \{g_\alpha^i\}_{i\in\mathcal{M}} \notin B_{n\alpha}.$$

Therefore, by Eq. (61), a.s. for large enough n,

$$\forall \alpha \in [n], \{g_\alpha^i\}_{i\in\mathcal{M}'} \notin A_{n\alpha}.$$

This finishes the proof of NullAvoid for $\mathcal{M}'$, and also CoreSet(m).

Lemma H.8. *Assume* Moments$(m-1)$. *Suppose* $\psi : \mathbb{R}^{m-1} \to \mathbb{R}$ *is controlled. Then as* $n \to \infty$,

$$\frac{1}{n^p} \max_{\alpha \in [n]} |\psi(g^1_\alpha, \ldots, g^{m-1}_\alpha)| \xrightarrow{\text{a.s.}} 0$$

for any $p > 0$.

Proof. For any $q > 0$, we have the elementary bound

$$\max_{\alpha \in [n]} |\psi(g^1_\alpha, \ldots, g^{m-1}_\alpha)| \le \sqrt[q]{\sum_{\alpha \in [n]} |\psi(g^1_\alpha, \ldots, g^{m-1}_\alpha)|^q}.$$

Thus, for any $q > 0$,

$$\frac{1}{n^p} \max_{\alpha \in [n]} |\psi(g^1_\alpha, \ldots, g^{m-1}_\alpha)| \le \frac{1}{n^{p-1/q}} \sqrt[q]{\frac{1}{n} \sum_{\alpha \in [n]} |\psi(g^1_\alpha, \ldots, g^{m-1}_\alpha)|^q}.$$

Because, by Moments$(m-1)$, $\frac{1}{n}\sum_{\alpha \in [n]} |\psi(g^1_\alpha, \ldots, g^{m-1}_\alpha)|^q \xrightarrow{\text{a.s.}} C$ for some constant C as $n \to \infty$, the RHS above converges a.s. to 0 as soon as we take $q > 1/p$, and therefore so does the LHS.

□

H.5 Inductive Step: Moments(m)

In this section, we show

$$\text{Moments}(m-1) \text{ and } \text{CoreSet}(m-1) \implies \text{Moments}(m).$$

More specifically, we will show that for any controlled $\psi : \mathbb{R}^m \to \mathbb{R}$,

$$\frac{1}{n} \sum_{\alpha=1}^{n} \psi(g^1_\alpha, \ldots, g^m_\alpha) \xrightarrow{\text{a.s.}} \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z)$$

where again on the RHS ψ ignores all coordinates $Z^{m+1}, \ldots, Z^M$ (corresponding to $g^{m+1}, \ldots, g^M$).

By Lemma H.7, if $\mathring{\sigma} = 0$, then almost surely, for large enough n, $g = g^m$ is just a (fixed) linear combination of $g^1, \ldots, g^{m-1}$, so Moments is trivially true. Therefore, in the below, we assume

$$\mathring{\sigma} > 0. \tag{$\star$}$$

This assumption will be crucial for our arguments involving smoothness induced by Gaussian averaging.

To clarify notation in the following, we will write $\mathbb{E}_X [expression]$ to denote the expectation over only the randomization in X, and $\mathbb{E}[expression|\mathcal{B}]$ to denote the expectation taken over all randomness except those in $\mathcal{B}$.

Proof Plan Note that

$$\left| \frac{1}{n} \sum_{\alpha=1}^{n} \psi(g^1_\alpha, \ldots, g^m_\alpha) - \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z) \right| \le \mathsf{A} + \mathsf{B} + \mathsf{C} \tag{62}$$

where

$$\mathsf{A} \stackrel{\text{def}}{=} \left| \frac{1}{n} \sum_{\alpha=1}^{n} \psi(g^1_\alpha, \ldots, g^m_\alpha) - \mathop{\mathbb{E}}_{z} \psi\left(g^1_\alpha, \ldots, g^{m-1}_\alpha, \omega_\alpha + \sigma z\right) \right|$$

$$\mathsf{B} \stackrel{\text{def}}{=} \left| \frac{1}{n} \sum_{\alpha=1}^{n} \mathop{\mathbb{E}}_{z} \psi\left(g^1_\alpha, \ldots, g^{m-1}_\alpha, \omega_\alpha + \sigma z\right) - \mathop{\mathbb{E}}_{z} \psi\left(g^1_\alpha, \ldots, g^{m-1}_\alpha, \sum_{i=1}^{r} \mathring{v}_i \hat{g}^i_\alpha + \mathring{\sigma} z\right) \right|$$

$$\mathsf{C} \stackrel{\text{def}}{=} \left| \frac{1}{n} \sum_{\alpha=1}^{n} \mathop{\mathbb{E}}_{z} \psi\left(g^1_\alpha, \ldots, g^{m-1}_\alpha, \sum_{i=1}^{r} \mathring{v}_i \hat{g}^i_\alpha + \mathring{\sigma} z\right) - \mathop{\mathbb{E}}_{Z \sim \mathcal{N}(\mu, \Sigma)} \psi(Z) \right|$$

with $z \sim \mathcal{N}(0,1)$. Note that B and C are random variables in $\mathcal{B}$. We will show that each of $\mathsf{A}, \mathsf{B}, \mathsf{C}$ goes to 0 almost surely, which would finish the proof of Theorem 5.4.

Roughly speaking, $\mathsf{A} \xrightarrow{\text{a.s.}} 0$ because of a law of large number, $\mathsf{B} \xrightarrow{\text{a.s.}} 0$ because of the smoothness in $\mathbb{E}_z \psi$ induced by Gaussian averaging, and $\mathsf{C} \xrightarrow{\text{a.s.}} 0$ by induction hypothesis. We start with the last item, since it's the easiest.

H.5.1 C Converges Almost Surely to 0

In this section we show that $\mathsf{C} \xrightarrow{\text{a.s.}} 0$ by a straightforward reduction to the inductive hypothesis.

Let $\hat{Z}^1, \ldots, \hat{Z}^r$ be the components of $Z \sim \mathcal{N}(\mu, \Sigma)$ corresponding to $\hat{g}^1, \ldots, \hat{g}^r$, and let $\hat{Z}$ be the column vector with these entries. Note that, by Proposition G.2, Z^m (corresponding to g^m), conditioned on $Z^1, \ldots, Z^{m-1}$, is distributed as a Gaussian with mean $\Sigma(g, \hat{G})\Sigma(\hat{G}, \hat{G})^+ \hat{Z} = \mathring{\hat{\eta}}^\top \mathring{\hat{\Lambda}}^+ \hat{Z} = \mathring{v}^\top \hat{Z}$ and variance $\Sigma(g,g) - \Sigma(g, \hat{G})\Sigma(\hat{G}, \hat{G})^+ \Sigma(\hat{G}, g) = \mathring{\sigma}$. Thus

$$\begin{aligned}
\mathbb{E}_{Z} \psi(Z) &= \mathbb{E}_{Z^1, \ldots, Z^{m-1}} \mathbb{E}[\psi(Z) | Z^1, \ldots, Z^{m-1}] \\
&= \mathbb{E}_{Z^1, \ldots, Z^{m-1}} \mathbb{E}_{z \sim \mathcal{N}(0,1)} \psi(Z^1, \ldots, Z^{m-1}, \mathring{v}^\top \hat{Z} + \mathring{\sigma} z) \\
&= \mathbb{E}_{Z^1, \ldots, Z^{m-1}} \Psi(Z^1, \ldots, Z^{m-1})
\end{aligned}$$

where we have set $\Psi(Z^1, \ldots, Z^{m-1}) \overset{\text{def}}{=} \mathbb{E}_{z \sim \mathcal{N}(0,1)} \psi(Z^1, \ldots, Z^{m-1}, \mathring{v}^\top \hat{Z} + \mathring{\sigma} z)$. Ψ is a controlled function since ψ is. Applying the induction hypothesis to Ψ, we obtain

$$\begin{aligned}
&\frac{1}{n} \sum_{\alpha=1}^{n} \mathbb{E}_{z} \psi \left(g_\alpha^1, \ldots, g_\alpha^{m-1}, \sum_{i=1}^{r} \mathring{v}_i \hat{g}_\alpha^i + \mathring{\sigma} z \right) \\
&= \frac{1}{n} \sum_{\alpha=1}^{n} \Psi \left(g_\alpha^1, \ldots, g_\alpha^{m-1} \right) \\
&\xrightarrow{\text{a.s.}} \mathbb{E}_{Z^1, \ldots, Z^{m-1}} \Psi(Z^1, \ldots, Z^{m-1}) \\
&\qquad\qquad\qquad\qquad \text{by induction hypothesis} \\
&= \mathbb{E}_{Z^1, \ldots, Z^{m-1}} \mathbb{E}_{z \sim \mathcal{N}(0,1)} \psi(Z^1, \ldots, Z^{m-1}, \mathring{v}^\top \hat{Z} + \mathring{\sigma} z) \\
&= \mathbb{E}_{Z} \psi(Z)
\end{aligned}$$

as desired.

H.5.2 A Converges Almost Surely to 0

In this section we show $\mathsf{A} \xrightarrow{\text{a.s.}} 0$ by a bounding moments of A and then finishing with Lemma G.1.

For each $\alpha \in [n]$, let $\psi_\alpha(x) \overset{\text{def}}{=} \psi(g_\alpha^1, \ldots, g_\alpha^{m-1}, \omega_\alpha + \sigma x)$, with ω and σ defined in Eq. (58). This is a random function depending on the randomness of $g_\alpha^1, \ldots, g_\alpha^{m-1}$, and it changes with n as well. Note by Eq. (59),

$$\mathsf{A} \overset{\text{d}}{=}_{\mathcal{B}} \frac{1}{n} \sum_{\alpha=1}^{n} \psi_\alpha(\xi_\alpha) - \mathbb{E}_{\xi'} \psi_\alpha(\xi'_\alpha)$$

where $\xi, \xi' \sim \mathcal{N}(0, I)$.

Since each summand $(\psi_\alpha(\xi_\alpha) - \mathbb{E}_{\xi'} \psi_\alpha(\xi'_\alpha))$ of A is independent from other summands when conditioned on $\mathcal{B}$, to show $\mathsf{A} \xrightarrow{\text{a.s.}} 0$, it suffices to bound the moments of each summand and then apply Lemma G.1. This is equivalent to bounding the uncentered moments $\mathbb{E}_{z \sim \mathcal{N}(0,1)} |\psi_\alpha(x)|^q$ for large enough q. Suppose ψ is λ-controlled and satisfies

$$|\psi(x)| \leq e^{C \sum_i |x_i|^\lambda + c} \qquad \text{for some } C, c > 0 \text{ and } \lambda < 2. \tag{63}$$

We have

$$\begin{aligned}\mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,1)}|\psi_\alpha(z)|^q &\leq \mathop{\mathbb{E}}_{z}\left[e^{Cq\left(|\omega_\alpha+\sigma z|^\lambda+\sum_{i=1}^{m-1}|g^i_\alpha|^\lambda\right)+cq}\right]\\ &\leq \mathop{\mathbb{E}}_{z}\left[e^{Cq2^\lambda\left(|\omega_\alpha|^\lambda+|\sigma z|^\lambda+\sum_{i=1}^{m-1}|g^i_\alpha|^\lambda\right)+cq}\right]\\ &= e^{Cq2^\lambda\left(|\omega_\alpha|^\lambda+\sum_{i=1}^{m-1}|g^i_\alpha|^\lambda\right)+cq}\mathop{\mathbb{E}}_{z}\left[e^{Cq2^\lambda\sigma^\lambda|z|^\lambda}\right]\\ &= e^{Cq2^\lambda\left(|\omega_\alpha|^\lambda+\sum_{i=1}^{m-1}|g^i_\alpha|^\lambda\right)+cq}R\end{aligned}$$

where $R=\mathsf{C}^1_\lambda(Cq2^\lambda\sigma^\lambda)>0$ is deterministic and C^k_λ is as defined in Lemma G.10. Now,

$$|\omega_\alpha|^\lambda=\left|\sum_{i=1}^r v_i\hat{g}^i_\alpha\right|^\lambda\leq r^\lambda\sum_{i=1}^r|v_i|^\lambda|\hat{g}^i_\alpha|^\lambda.$$

Additionally, almost surely, $|v_i|<|\mathring{v}_i|+1$, for all $i\in[r]$ simultaneously, for large enough n because $v_i\xrightarrow{\text{a.s.}}\mathring{v}_i$. Let $L=Cq2^\lambda r^\lambda\max_{i=1}^r(|\mathring{v}_i|+1)$ and $L'=cq$, where C,c are as in Eq. (63). Then, almost surely, for large enough n, for all $z^1,\dots,z^{m-1}\in\mathbb{R}$,

$$e^{Cq2^\lambda\left(\left|\sum_{i=1}^r v_iz^i\right|^\lambda+\sum_{i=1}^{m-1}|z^i|^\lambda\right)+cq}\leq e^{L'+L\sum_{i=1}^{m-1}|z^i|^\lambda}\overset{\text{def}}{=}\hat{\psi}(z^1,\dots,z^{m-1}).$$

Obviously $\hat{\psi}$ is λ-controlled. Then, again a.s. for large enough n, simultaneously for all α,

$$\mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,1)}|\psi_\alpha(z)|^q\leq R\hat{\psi}(g^1_\alpha,\dots,g^{m-1}_\alpha),\quad\text{so that}$$
$$\frac{1}{n}\sum_{\alpha=1}^n\mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,1)}|\psi_\alpha(z)|^q\leq R\frac{1}{n}\sum_{\alpha=1}^n\hat{\psi}(g^1_\alpha,\dots,g^{m-1}_\alpha)\xrightarrow{\text{a.s.}}R\mathop{\mathbb{E}}_{Z}\hat{\psi}(Z)$$

as $n\to\infty$, by induction hypothesis, where $Z\sim\mathcal{N}(\mu,\Sigma)$. Consequently, almost surely, the moment $\mathbb{E}[|\mathsf{A}|^q\mid\mathcal{B}]$ (as a function of $g^1,\dots,g^{m-1}$) is uniformly bounded in n. Applying Lemma G.1 for large enough $q\in\mathbb{N}$ yields the result.

H.5.3 B Converges Almost Surely to 0

In this section we show $\mathsf{B}\xrightarrow{\text{a.s.}}0$. The main insight here is integrating a function against Gaussian induces smoothness in the function. We will assume that $\mathring{\sigma}>0$, so that $\sigma>0$ almost surely for large enough n. This is because $\mathring{\sigma}=0$ implies that g^m is in the linear span of $\{g^1,\dots,g^{m-1}\}$ almost surely by Lemma H.4, and Moments(m) then holds trivially.

For each $\alpha\in[n]$, $w\in\mathbb{R}$, $\tau\geq0$, let

$$\Psi_\alpha(w;\tau^2)\overset{\text{def}}{=}\mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,1)}\psi\left(g^1_\alpha,\dots,g^{m-1}_\alpha,w+\tau z\right).$$

(Here and in all that follows, τ^2 is the square of τ, and the 2 is not an index). This is a random function, with randomness induced by $g^1,\dots,g^{m-1}$.

By Lemma G.3, Ψ_α is differentiable in w, and

$$\partial_w\Psi_\alpha(w;\tau^2)=\tau^{-1}\mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,1)}z\psi(g^1_\alpha,\dots,g^{m-1}_\alpha,w+\tau z).$$

We can obtain the following smoothness condition on Ψ_α.

Lemma H.9. *For any $w,\tau,\epsilon\in\mathbb{R}$ with $\epsilon,\tau>0$,*

$$|\Psi_\alpha(w;\tau^2)-\Psi_\alpha(w+\epsilon;\tau^2)|\leq|\epsilon|\tau^{-1}R(\tau)\hat{\Psi}(g^1_\alpha,\dots,g^{m-1}_\alpha)e^{C4^\lambda(|w|^\lambda+|\epsilon|^\lambda)},$$

where $\hat{\Psi}(g^1_\alpha,\dots,g^{m-1}_\alpha)\overset{\text{def}}{=}e^{C2^\lambda\sum_{i=1}^{m-1}|g^i_\alpha|^\lambda+c}$ and $R(\tau)\overset{\text{def}}{=}\mathbb{E}_z|z|e^{C2^\lambda\tau^\lambda|z|^\lambda}$.

Proof. Clearly, with $z \sim \mathcal{N}(0,1)$,

$$\begin{aligned}
|\partial_w \Psi_\alpha(w;\tau^2)| &\le \tau^{-1} \mathop{\mathbb{E}}_z |z\psi(g^1_\alpha, \dots, g^{m-1}_\alpha, w + \tau z)| \\
&\le \tau^{-1} \mathop{\mathbb{E}}_z |z| e^{C\left(|w+\tau z|^\lambda + \sum_{i=1}^{m-1} |g^i_\alpha|^\lambda\right)+c} \\
&\le \tau^{-1} \mathop{\mathbb{E}}_z |z| e^{C2^\lambda\left(|w|^\lambda + \tau^\lambda |z|^\lambda + \sum_{i=1}^{m-1} |g^i_\alpha|^\lambda\right)+c} \\
&= \tau^{-1} \hat{\Psi}(g^1_\alpha, \dots, g^{m-1}_\alpha) R(\tau) e^{C2^\lambda |w|^\lambda}.
\end{aligned}$$

Then

$$\begin{aligned}
&|\Psi_\alpha(w;\tau^2) - \Psi_\alpha(w+\epsilon;\tau^2)| \\
&\le \left| \int_w^{w+\epsilon} \mathrm{d}\xi \; \partial_\xi \Psi_\alpha(\xi;\tau^2) \right| \\
&\le \tau^{-1} R(\tau) \hat{\Psi}(g^1_\alpha, \dots, g^{m-1}_\alpha) \int_w^{w+\epsilon} \mathrm{d}\xi \; e^{C2^\lambda |\xi|^\lambda} \\
&= \tau^{-1} R(\tau) \hat{\Psi}(g^1_\alpha, \dots, g^{m-1}_\alpha) \int_0^{\epsilon} \mathrm{d}\xi \; e^{C2^\lambda |w+\xi|^\lambda} \\
&\le \tau^{-1} R(\tau) \hat{\Psi}(g^1_\alpha, \dots, g^{m-1}_\alpha) \int_0^{\epsilon} \mathrm{d}\xi \; e^{C4^\lambda |w|^\lambda} e^{C4^\lambda |\xi|^\lambda} \\
&= \tau^{-1} R(\tau) \hat{\Psi}(g^1_\alpha, \dots, g^{m-1}_\alpha) e^{C4^\lambda |w|^\lambda} |\epsilon| e^{C4^\lambda |\epsilon|^\lambda}.
\end{aligned}$$

□

Therefore, with $z \sim \mathcal{N}(0,1)$,

$$\begin{aligned}
&\left| \mathop{\mathbb{E}}_z \left[\psi(g^1_\alpha, \dots, g^{m-1}_\alpha, \sum_{i=1}^r v_i \hat{g}^i_\alpha + \sigma z) \right] - \mathop{\mathbb{E}}_z \left[\psi(g^1_\alpha, \dots, g^{m-1}_\alpha, \sum_{i=1}^r \mathring{v}_i \hat{g}^i_\alpha + \sigma z) \right] \right| \\
&= \left| \Psi_\alpha\left(\sum_{i=1}^r v_i \hat{g}^i_\alpha ; \sigma^2 \right) - \Psi_\alpha\left(\sum_{i=1}^r \mathring{v}_i \hat{g}^i_\alpha ; \sigma^2 \right) \right| \\
&\le \sigma^{-1} R(\sigma) \hat{\Psi}(g^1_\alpha, \dots, g^{m-1}_\alpha) e^{C4^\lambda \left(\left|\sum_{i=1}^r \mathring{v}_i \hat{g}^i_\alpha\right|^\lambda + |\epsilon_\alpha|^\lambda \right)} |\epsilon_\alpha|,
\end{aligned}$$

where $\epsilon_\alpha \overset{\text{def}}{=} \sum_{i=1}^r (v_i - \mathring{v}_i) \hat{g}^i_\alpha$. Because $\hat{\Psi}$ is λ-controlled,

$$\frac{1}{n} \sum_{\alpha=1}^n \hat{\Psi}(g^1_\alpha, \dots, g^{m-1}_\alpha)$$

converges almost surely to a deterministic limit. At the same time, since $v_i \xrightarrow{\text{a.s.}} \mathring{v}_i$, we also have $\epsilon_\alpha \xrightarrow{\text{a.s.}} 0$ as $n \to \infty$, so that

$$\frac{1}{n} \sum_{\alpha=1}^n \left| \mathop{\mathbb{E}}_z \left[\psi(g^1_\alpha, \dots, g^{m-1}_\alpha, \sum_{i=1}^r v_i \hat{g}^i_\alpha + \sigma z) \right] - \mathop{\mathbb{E}}_z \left[\psi(g^1_\alpha, \dots, g^{m-1}_\alpha, \sum_{i=1}^r \mathring{v}_i \hat{g}^i_\alpha + \sigma z) \right] \right| \xrightarrow{\text{a.s.}} 0.$$

A similar argument shows that we can replace σ with $\mathring{\sigma}$:

$$\frac{1}{n} \sum_{\alpha=1}^n \left| \mathop{\mathbb{E}}_z \left[\psi(g^1_\alpha, \dots, g^{m-1}_\alpha, \sum_{i=1}^r \mathring{v}_i \hat{g}^i_\alpha + \sigma z) \right] - \mathop{\mathbb{E}}_z \left[\psi(g^1_\alpha, \dots, g^{m-1}_\alpha, \sum_{i=1}^r \mathring{v}_i \hat{g}^i_\alpha + \mathring{\sigma} z) \right] \right| \xrightarrow{\text{a.s.}} 0.$$

By triangular inequality, these limits show that $\mathsf{B} \xrightarrow{\text{a.s.}} 0$ as desired.

I Proof of NETSOR$^+$ Master Theorem

In this section we describe how to augment the proof of Theorem E.3 given in Appendix H to yield the proof of Theorem C.4. The key points to note here are 1) the presence of `LinComb` rules in NETSOR$^+$ but not in NETSOR$^-$, 2) the rank stability assumption Assumption C.3 used in Theorem C.4, and 3) an additional term in Eq. (62) due to fluctuations in the parameter Θ.

I.1 `LinComb`

As remarked in Remark E.2, any usage of `LinComb` in a NETSOR$^+$ program can be absorbed into downstream nonlinearities or be expressed as `Nonlin`$^+$ rule. So WLOG, we can assume that the NETSOR$^+$ program has no applications of `LinComb`.

I.2 Rank Stability

By Remark C.6, we see that rank stability assumption is necessary for the NETSOR$^+$ Master Theorem. Whereas in Appendix H, we had to intricately weave together an induction on rank stability (more generally, CoreSet) and an induction on moment convergence (Moments), here to show Theorem C.4, we just need 1) induct on Moments and 2) to invoke Assumption C.3 whenever we need to use Lemma H.4, which is when we need to show that pseudo-inverse commutes with almost surely limit, such as in Proposition H.1, and when we need to ensure either σ is almost surely 0 or is almost surely positive, as in Appendix H.5.3.

I.3 Fluctuation of the Parameters

When we have parameters in nonlinearities, Eq. (62) needs to be modified to contain an additional term D:

$$\left|\frac{1}{n}\sum_{\alpha=1}^{n}\psi(g^1_\alpha,\ldots,g^m_\alpha;\Theta)-\mathop{\mathbb{E}}_{Z\sim\mathcal{N}(\mu,\Sigma)}\psi(Z;\mathring{\Theta})\right|\le\mathsf{D}+\mathsf{A}+\mathsf{B}+\mathsf{C}$$

where

$$\mathsf{D}\overset{\text{def}}{=}\left|\frac{1}{n}\sum_{\alpha=1}^{n}\psi(g^1_\alpha,\ldots,g^m_\alpha;\Theta)-\psi(g^1_\alpha,\ldots,g^m_\alpha;\mathring{\Theta})\right|$$

and $\mathsf{A},\mathsf{B},\mathsf{C}$ are as in Eq. (62) but replacing $\psi(-)$ there with $\psi(-;\mathring{\Theta})$. Because $\psi(-;-)$ is parameter-controlled at $\mathring{\Theta}$ by assumption, $\psi(-;\mathring{\Theta})$ is controlled, and $\mathsf{A},\mathsf{B},\mathsf{C}\xrightarrow{\text{a.s.}}0$ with the same arguments as before (except using rank stability assumption Assumption C.3 where appropriate, instead of CoreSet).

Now, by the other property of parameter-control, we have

$$\begin{aligned}\mathsf{D}&\le\frac{1}{n}\sum_{\alpha=1}^{n}\left|\psi(g^1_\alpha,\ldots,g^m_\alpha;\Theta)-\psi(g^1_\alpha,\ldots,g^m_\alpha;\mathring{\Theta})\right|\\&\le\frac{1}{n}\sum_{\alpha=1}^{n}f(\Theta)\bar{\psi}(g^1_\alpha,\ldots,g^m_\alpha)\\&=f(\Theta)\frac{1}{n}\sum_{\alpha=1}^{n}\bar{\psi}(g^1_\alpha,\ldots,g^m_\alpha)\end{aligned}$$

for some controlled $\bar{\psi}:\mathbb{R}^m\to\mathbb{R}$ and some $f:\mathbb{R}^l\to\mathbb{R}^{\ge0}\cup\{\infty\}$ that is continuous at $\mathring{\Theta}$ and has $f(\mathring{\Theta})=0$ (where $\bar{\psi}$ and f can both depend on $\mathring{\Theta}$). Since $\Theta\xrightarrow{\text{a.s.}}\mathring{\Theta}$, we have $f(\Theta)\xrightarrow{\text{a.s.}}0$. In addition, by Moments, $\frac{1}{n}\sum_{\alpha=1}^{n}\bar{\psi}(g^1_\alpha,\ldots,g^m_\alpha)$ converges to a.s. as well to a finite constant. Therefore,

$$\mathsf{D}\xrightarrow{\text{a.s.}}0$$

as desired.

I.4 Summary

The proof of Theorem C.4, WLOG for programs without `LinComb`, would proceed as follows: We induct on Moments with the same setup as Appendix H.2, except using Assumption C.3 for Proposition H.1. Then we prove the inductive step for Moments as in Appendix H.5. We modify Eq. (62) to add a term D as in Appendix I.3, which goes to 0 a.s. as argued there. The same arguments for $\mathsf{A},\mathsf{B},\mathsf{C}\xrightarrow{\text{a.s.}}0$, exhibited in Appendix H.5 still hold, except that in the proof of $\mathsf{B}\xrightarrow{\text{a.s.}}0$, we apply Assumption C.3 (instead of Lemma H.4) to allow us to assume $\mathring{\sigma}>0$ and $\sigma>0$ almost surely.

```
program ::= stmt*

stmt ::= Input var :: type
       | var := expr :: type

expr ::= MatMul (var, var )
       | fun( var* )

var ::= ⟨ id ⟩

fun ::= ⟨ function ℝ^k → ℝ for some k ≥ 0 ⟩

type ::= G(nat) | H(nat) | A(nat, nat)

nat ::= ⟨ any integer ≥ 1 ⟩
```

Figure 4: NETSOR$^-$ Grammar; see Definition E.1.

$$\frac{\mathsf{expr} : \mathsf{type} \qquad \mathsf{var} \mathbin{\mathbf{:=}} \mathsf{expr} \mathbin{\mathbf{::}} \mathsf{type}}{\mathsf{var} : \mathsf{type}}$$

$$\frac{\mathsf{a} : \mathbf{A}(n_1, n_2) \qquad \mathsf{h} : \mathbf{H}(n_2)}{\mathbf{MatMul}(\mathsf{a}, \mathsf{h}) : \mathbf{G}(n_1)} \qquad \frac{\mathsf{g}_1, \ldots, \mathsf{g}_k : \mathbf{G}(n) \qquad \mathsf{f} : \mathbb{R}^k \to \mathbb{R}}{\mathsf{f}(\mathsf{g}_1, \ldots, \mathsf{g}_k) : \mathbf{H}(n)}$$

Figure 5: NETSOR$^-$ Inference Rules

J Formal Specification of Tensor Programs

In the main text, we have adopted an informal approach to specifying the NETSOR language and its siblings, in order to make the material accessible to a wide audience. Here we give the formal specifications for NETSOR$^-$ (Figs. 4 to 6), NETSOR (Figs. 7 to 9), and self-parametrized NETSOR$^+$ (Figs. 10 to 12). For ease of presentation, we have represented matrix multiplication explicitly via an operation **MatMul** (likewise for **Moment** in self-parametrized NETSOR$^+$), and have we used double colon :: instead of single colon : for type annotation.

$$\frac{[\![\mathsf{a}]\!] = W \in \mathbb{R}^{n_1 \times n_2} \qquad [\![\mathsf{h}]\!] = v \in \mathbb{R}^{n_2}}{[\![\mathbf{MatMul}(\mathsf{a}, \mathsf{h})]\!] = Wv} \qquad \frac{\forall i \in [k], [\![\mathsf{g}_i]\!] = v_i \in \mathbb{R}^n \qquad [\![\mathsf{f}]\!] = f : \mathbb{R}^k \to \mathbb{R}}{[\![\mathsf{f}(\mathsf{g}_1, \ldots, \mathsf{g}_k)]\!] = u \in \mathbb{R}^n \text{ with } u_\alpha = f(v_{1\alpha}, \ldots, v_{k\alpha})}$$

Figure 6: NETSOR$^-$ Semantics

```
program ::= stmt*

stmt ::= Input var :: type
       | var := expr :: type

expr ::= MatMul (var, var )
       | fun( var* )
       | var (+ var)+

var ::= ⟨ id ⟩

fun ::= ⟨ function ℝ^k → ℝ for some k ≥ 0 ⟩

type ::= G(nat) | H(nat) | A(nat, nat)

nat ::= ⟨ any integer ≥ 1 ⟩
```

Figure 7: NETSOR Grammar; see Definition 4.1. Compared to NETSOR$^-$ grammar, the only new item is `LinComb` in *expr*.

$$\frac{\mathsf{expr} : \mathsf{type} \qquad \mathsf{var} \mathbin{\textbf{:=}} \mathsf{expr} \mathbin{\textbf{::}} \mathsf{type}}{\mathsf{var} : \mathsf{type}}$$

$$\frac{\mathsf{a} : \mathbf{A}(n_1, n_2) \qquad \mathsf{h} : \mathbf{H}(n_2)}{\mathbf{MatMul}(\mathsf{a}, \mathsf{h}) : \mathbf{G}(n_1)} \qquad \frac{\mathsf{g}_1, \ldots, \mathsf{g}_k : \mathbf{G}(n) \qquad \mathsf{f} : \mathbb{R}^k \to \mathbb{R}}{\mathsf{f}(\mathsf{g}_1, \ldots, \mathsf{g}_k) : \mathbf{H}(n)} \qquad \frac{\mathsf{g}_1, \ldots, \mathsf{g}_k : \mathbf{G}(n)}{\mathsf{g}_1 + \cdots + \mathsf{g}_k : \mathbf{G}(n)}$$

Figure 8: NETSOR Inference Rules

$$\frac{[\![\mathsf{a}]\!] = W \in \mathbb{R}^{n_1 \times n_2} \qquad [\![\mathsf{h}]\!] = v \in \mathbb{R}^{n_2}}{[\![\mathbf{MatMul}(\mathsf{a}, \mathsf{h})]\!] = Wv} \qquad \frac{\forall i \in [k], [\![\mathsf{g}_i]\!] = v_i \in \mathbb{R}^n \qquad [\![\mathsf{f}]\!] = f : \mathbb{R}^k \to \mathbb{R}}{[\![\mathsf{f}(\mathsf{g}_1, \ldots, \mathsf{g}_k)]\!] = u \in \mathbb{R}^n \text{ with } u_\alpha = f(v_{1\alpha}, \ldots, v_{k\alpha})}$$

$$\frac{\forall i \in [k], [\![\mathsf{g}_i]\!] = v_i \in \mathbb{R}^n}{[\![\mathsf{g}_1 + \cdots + \mathsf{g}_k]\!] = v_1 + \cdots + v_k \in \mathbb{R}^n}$$

Figure 9: NETSOR Semantics

```
program ::= stmt*

stmt ::= Input var :: type
       | var := expr :: type

expr ::= MatMul (var, var )
       | fun( var* ; var*)
       | var (+ var)+
       | Moment(fun; var*; var*)

var ::= ⟨ id ⟩

fun ::= ⟨ parametrized function ℝ^k × ℝ^l → ℝ for some k, l ≥ 0 ⟩

type ::= C | G(nat) | H(nat) | A(nat, nat)

nat ::= ⟨ any integer ≥ 1 ⟩
```

Figure 10: Self-Parametrized NETSOR$^+$ Grammar; see Definition C.8. Compared to NETSOR grammar, we have added a new type **C** and a new expression **Moment**.

$$\frac{\mathsf{expr} : \mathsf{type} \qquad \mathsf{var}\ \mathbf{:=}\ \mathsf{expr}\ \mathbf{::}\ \mathsf{type}}{\mathsf{var} : \mathsf{type}}$$

$$\frac{\mathsf{a} : \mathbf{A}(n_1, n_2) \qquad \mathsf{h} : \mathbf{H}(n_2)}{\mathbf{MatMul}(\mathsf{a}, \mathsf{h}) : \mathbf{G}(n_1)} \qquad \frac{\mathsf{g}_1, \ldots, \mathsf{g}_k : \mathbf{G}(n)}{\mathsf{g}_1 + \cdots + \mathsf{g}_k : \mathbf{G}(n)}$$

$$\frac{\mathsf{g}_1, \ldots, \mathsf{g}_k : \mathbf{G}(n) \qquad \mathsf{c}_1, \ldots, \mathsf{c}_l : \mathbf{C} \qquad \mathsf{f} : \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}}{\mathsf{f}(\mathsf{g}_1, \ldots, \mathsf{g}_k; \mathsf{c}_1, \ldots, \mathsf{c}_l) : \mathbf{H}(n)}$$

$$\frac{\mathsf{g}_1, \ldots, \mathsf{g}_k : \mathbf{G}(n) \qquad \mathsf{c}_1, \ldots, \mathsf{c}_l : \mathbf{C} \qquad \mathsf{f} : \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}}{\mathbf{Moment}(\mathsf{f}; \mathsf{g}_1, \ldots, \mathsf{g}_k; \mathsf{c}_1, \ldots, \mathsf{c}_l) : \mathbf{C}}$$

Figure 11: Self-Parametrized NETSOR$^+$ Inference Rules

$$\frac{[\![\mathsf{a}]\!] = W \in \mathbb{R}^{n_1 \times n_2} \qquad [\![\mathsf{h}]\!] = v \in \mathbb{R}^{n_2}}{[\![\mathbf{MatMul}(\mathsf{a}, \mathsf{h})]\!] = Wv} \qquad \frac{\forall i \in [k], [\![\mathsf{g}_i]\!] = v_i \in \mathbb{R}^n}{[\![\mathsf{g}_1 + \cdots + \mathsf{g}_k]\!] = v_1 + \cdots + v_k \in \mathbb{R}^n}$$

$$\frac{\forall i \in [k], [\![\mathsf{g}_i]\!] = v_i \in \mathbb{R}^n \qquad \forall j \in [l], [\![\mathsf{c}_j]\!] = c_j \in \mathbb{R} \qquad [\![\mathsf{f}]\!] = f : \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}}{[\![\mathsf{f}(\mathsf{g}_1, \ldots, \mathsf{g}_k; \mathsf{c}_1, \ldots, \mathsf{c}_l)]\!] = u \in \mathbb{R}^n \text{ with } u_\alpha = f(v_{1\alpha}, \ldots, v_{k\alpha}; c_1, \ldots, c_l)}$$

$$\frac{\forall i \in [k], [\![\mathsf{g}_i]\!] = v_i \in \mathbb{R}^n \qquad \forall j \in [l], [\![\mathsf{c}_j]\!] = c_j \in \mathbb{R} \qquad [\![\mathsf{f}]\!] = f : \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}}{[\![\mathbf{Moment}(\mathsf{f}; \mathsf{g}_1, \ldots, \mathsf{g}_k; \mathsf{c}_1, \ldots, \mathsf{c}_l)]\!] = \frac{1}{n}\sum_{\alpha=1}^{n} f(v_{1\alpha}, \ldots, v_{k\alpha}; c_1, \ldots, c_l)}$$

Figure 12: Self-Parametrized NETSOR$^+$ Semantics