Correlated Weights in Infinite Limits of Deep Convolutional Neural Networks (Supplementary material)

Adrià Garriga-Alonso[1] **Mark van der Wilk**[2]

[1]Department of Engineering, University of Cambridge, UK
[2]Department of Computer Science, Imperial College London, UK

A PATCH FUNCTIONS AND DISCRETE CONVOLUTIONS

Usually, convolutions are defined explicitly by subtracting the indices of one input tensor from the other one, and not using patch functions. To make this paper clearer, it is convenient to abstract the details of a convolution, so we introduced the patch function.

Definition A.1 (Discrete convolution)**.** Let $D \in \mathbb{N}$ be a number of spatial dimensions, the tensor-valued weights $\boldsymbol{W} \in \mathbb{R}^{\boldsymbol{P}}$, input $\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{F}}$, and output $\boldsymbol{Y} \in \mathbb{R}^{\boldsymbol{F}'}$. The tensor sizes $\boldsymbol{P}$ (patch size) and $\boldsymbol{F}, \boldsymbol{F}'$ (feature sizes) are each a D-tuple, $\boldsymbol{P}, \boldsymbol{F}, \boldsymbol{F}' \in \mathbb{N}^D$. We say that $\boldsymbol{Y}$ is the result of the convolution operation $\boldsymbol{Y} = \boldsymbol{W} * \boldsymbol{X}$, if

$$Y_{q_1,\ldots,q_D} = \sum_{p_1=1}^{P_1} \cdots \sum_{p_D=1}^{P_D} W_{p_1,\ldots,p_D} X_{\tilde{q}_1(p_1),\ldots,\tilde{q}_D(p_D)}. \tag{1}$$

Here, $\tilde{q}_d(\cdot) : [P_d] \to [F'_d]$ are the *patch functions* for a given output location $\boldsymbol{q}$. Using D-tuples $\boldsymbol{p}, \boldsymbol{q}$ as indices, we may also write

$$Y_{\boldsymbol{q}} = \sum_{\boldsymbol{p}=\boldsymbol{1}}^{\boldsymbol{P}} W_{\boldsymbol{p}} X_{\tilde{\boldsymbol{q}}(\boldsymbol{p})}. \tag{2}$$

Counting from $\boldsymbol{1}$ to $\boldsymbol{P}$ is done in such a way that $\boldsymbol{p}$ takes all the values in $[\boldsymbol{P}]$.

Definition A.2 (Patch function)**.** For each dimension $d \in [D]$, layer $\ell \in [L]$, fix a stride $s \in \mathbb{N}$, and dilation $h \in \mathbb{N}$. For output position $\boldsymbol{q}$, the patch function of the dth dimension $\tilde{q}_d(\cdot) : [P_d] \to [F'_d]$ is

$$\tilde{q}_d(p_d) = sq_d - h\left(p_d - \left\lceil \frac{P_d}{2} \right\rceil\right). \tag{3}$$

For a D-tuple index $\boldsymbol{p}$, we may compactly write $\tilde{\boldsymbol{q}}(\boldsymbol{p}) \triangleq (\tilde{q}_1(p_1), \ldots, \tilde{q}_D(p_D))$.

It is possible to verify that definition A.2 overall yields the usual definition of a convolution in deep learning [Goodfellow et al., 2016, Section 9.1].

Remark A.3. The concatenation of two patch functions $\tilde{\boldsymbol{q}}(\cdot), \tilde{\boldsymbol{q}}'(\cdot)$ is also a patch function, with argument in $[\boldsymbol{P}]^2$. That is, for $[\boldsymbol{p}, \boldsymbol{p}'] = \boldsymbol{s} \in \mathbb{N}^{\boldsymbol{P}\times\boldsymbol{P}}$ and $[\boldsymbol{q}, \boldsymbol{q}'] = \boldsymbol{r}$,

$$\begin{aligned}&(\tilde{\boldsymbol{q}}(\boldsymbol{p}), \tilde{\boldsymbol{q}}'(\boldsymbol{p}')) \\ &\quad = (\tilde{q}_1(p_1), \ldots, \tilde{q}_D(p_D), \tilde{q}'_1(p'_1), \ldots, \tilde{q}'_D(p'_D)) \\ &\quad = \tilde{\boldsymbol{r}}(\boldsymbol{s})\,.\end{aligned} \tag{4}$$

B PROOF THAT A CNN WITH CORRELATIONS IN THE WEIGHTS CONVERGES TO A GP

In this section, we formally prove that a CNN with correlated weights converges in distribution to a Gaussian process in the limit of infinite width. Using the NETSOR programming language due to Yang [2019], most of the work in the proof is done by one step: describe a CNN with correlated weights in NETSOR.

For the reader's convenience, we informally recall the NETSOR programming language [Yang, 2019] and key properties of its programs (theorem B.7 and corollary B.8). The outline of our presentation here also closely follows Yang [2019]. Readers familiar with NETSOR should skip to appendix B.3, where we show the program that proves theorem B.10.

B.1 DEFINITION OF A NETSOR PROGRAM

A NETSOR program expresses numerical computations, such as those used to define the output of a neural network. Each line of a NETSOR program is simply the definition of a new variable, in terms of previously defined variables.

There are three types of variables: $\mathsf{G}(n)$-vars, $\mathsf{A}(n_1, n_2)$-vars, and $\mathsf{H}(n)$-vars (henceforth called "NETSOR variables"). Each of these have one or two parameters, which are the widths we will take to infinity. For a given index in $[n]$ (or $[n_1] \times [n_2]$), each NETSOR variable is a random

scalar. To represent vectors that do not grow to infinity, we need to use collections of NETSOR variables.

G-vars, A-vars and H-vars are all random when the program is run. To accomodate non-random variables that may change (like the input $\boldsymbol{X}$ to a neural network (NN)) we must define a different NETSOR program, defining $\boldsymbol{X}$ either as a constant or a G-var with variance zero.

What follows is an explanation of the three kinds of NETSOR variables, and example uses of them. The program indicates the type of a variable using "var : Type".

G-vars (Gaussian-vars) are n-wise *approximately* independent and identically distributed (i.i.d.) and Gaussian. By "n-wise (approximately) independent" we mean that there can be correlations between G-vars, but only within a single index $i \in 1, \ldots, n$. G-vars will converge in distribution to an n-wise independent, identically distributed Gaussian in the limit of $n \to \infty$, if all widths are n. They are used, for example, to define the biases of a fully connected neural network (FCNN).

A-vars represent matrices, like the weight matrices of a dense neural network. Their entries are always i.i.d. Gaussian with with zero mean, even for finite instantiations of the program (finite n). There are no correlations between different A-vars, or elements of the same A-var. They may be used to define the weight matrices of a FCNN.

H-vars represent variables that become n-wise i.i.d. (not necessarily Gaussian) in the infinite limit. G is a subtype of H, so all G-vars are also H-vars. Post-nonlinearity activations are H-vars.

O-vars (Output-vars) are used to define the output of the NETSOR program. A $\mathsf{O}(n)$-var behaves like you would expect a hypothetical $\mathsf{A}(n, 1)$-var to behave: its elements are i.i.d. Gaussian with mean zero, and it is independent of all other variables in the program.

Yang [2019] does not define O-vars, instead choosing to consider them part of the G-vars, since they both converge to i.i.d. Gaussians.

Definition B.1 (Netsor program)**.** A NETSOR program consists of:

Input: A set that may contain G-vars, A-vars, and O-vars.

Body: Each line of the program defines a new variable in terms of existing ones. New variables may be defined using the following rules:

- `MatMul`: $\mathsf{A}(n_1, n_2) \times \mathsf{H}(n_2) \to \mathsf{G}(n_1)$. Given an $\mathsf{A}(n_1, n_2)$-var (i.i.d. Gaussian matrix) and an $\mathsf{H}(n_2)$-var (i.i.d. vector), its multiplication is a $\mathsf{G}(n_1)$-var (that is, it converges to a Gaussian vector in the limit $n_2 \to \infty$).
- `LinComb`: Given constants $\alpha_1, \ldots, \alpha_K$, and G-vars $x_1, \ldots, x_K$ of type $\mathsf{G}(n_1)$, their linear combination $\sum_{k=1}^{K} \alpha_k x_k$ is a G-var.
- `Nonlin`: applying an elementwise nonlinear function $\phi : \mathbb{R}^K \to \mathbb{R}$, we map several G-vars $x_1, \ldots, x_K$ to one H-var.

Output: A tuple of scalars $(o_1^\mathsf{T} x_1, \ \ldots, \ o_K^\mathsf{T} x_K / \sqrt{n_K})$. The variables $o_k : \mathsf{O}(n_k)$ are O-vars. It may be the case that $o_j = o_k$ for different j, k (that is, the list $[v_1, \ldots, v_K]$ has repeated entries). Each $x_k : \mathsf{H}(n_k)$ is a H-var.

Outside of these rules, NETSOR does not have conditionals or loops[1]. In practice, we may use loops and conditionals to write a NETSOR program, so long as they do not access the values of NETSOR variables. Conceptually, these behave like a LISP-style "macro" that generates a NETSOR program.

B.2 THE OUTPUT OF A NETSOR PROGRAM CONVERGES TO A GAUSSIAN PROCESS

For simplicity, we assume that the width of all the NETSOR variables is n. Yang [2019] also considers the case where each n_k is different. First, the necessary assumptions.

Definition B.2 (Controlled function [Yang, 2019])**.** A function $\phi : \mathbb{R}^k \to \mathbb{R}$ is *controlled* if it is measurable and

$$|\phi(\boldsymbol{x})| \leq \exp\Big(C\|\boldsymbol{x}\|_2^{(2-\epsilon)}\Big) + c$$

for some $C, c, \epsilon > 0$, where $\|\cdot\|_2$ is the L2 norm.

If a function is controlled, it is L2 integrable with a Gaussian. That is, if the argument x of the function is Gaussian, the variance of $\phi(x)$ is finite. This in turn ensures that the NN function has finite variance. All common nonlinearities (ReLU, tanh, SiLU, ...) are controlled. This is a very weak assumption, it is vanishingly unlikely that future nonlinearities will grow as fast as $O\Big(e^{x^2}\Big)$.

Assumption B.3. All nonlinear functions $\phi(\cdot)$ in the NETSOR program are controlled.

Assumption B.4 (Distribution of A-var inputs)**.** Consider each $\mathsf{A}(n, n)$-var in the program, $\boldsymbol{W}$. Each of its elements $W_{i,j}$, where $i, j \in [n]$, is sampled from the zero-mean, i.i.d. Gaussian, $W_{i,j} \overset{iid}{\sim} \mathcal{N}\big(0, \sigma_\mathrm{w}^2/n\big)$.

Assumption B.5 (Distribution of G-var inputs)**.** Consider the input vector of all $\mathsf{G}(n)$-vars for each channel $i \in [n]$, that is the vector $\boldsymbol{z}_i \triangleq [x_i : x \text{ is input G-var}]$. It is drawn from a Gaussian, $\boldsymbol{z}_i \overset{iid}{\sim} \mathcal{N}\big(\boldsymbol{\mu}^\text{in}, \boldsymbol{\Sigma}^\text{in}\big)$. The covariance $\boldsymbol{\Sigma}^\text{in}$ may be singular.

[1] Of course, a nonlinearity ϕ may be internally defined using loops and conditionals, so long as it satisfies assumption B.3.

Assumption B.6 (Distribution of O-vars)**.** Each $\mathsf{O}(n_k)$-var v_k in the program is an independent Gaussian for each channel. Different O-vars may have different variances. That is, for each $k \in [K], i \in [n]$, $v_{k,i} \overset{iid}{\sim} \mathcal{N}\big(0, \sigma_k^2/n_k\big)$.

Theorem B.7 (NETSOR master theorem, Yang, 2019)**.** *Fix any* NETSOR *program satisfying assumptions B.3 to B.6. If $g^{(1)}, \dots, g^{(M)}$ are all the* G*-vars in the entire program, then for any controlled $\psi : \mathbb{R}^M \to \mathbb{R}$, as $n \to \infty$,*

$$\frac{1}{n}\sum_{i=1}^{n} \psi\Big(g_i^{(1)}, \dots, g_i^{(M)}\Big) \overset{a.s.}{\to} \mathop{\mathbb{E}}_{\boldsymbol{z}\sim\mathcal{N}(\boldsymbol{m},\boldsymbol{K})} \Big[\psi\Big(z^{(1)}, \dots, z^{(M)}\Big)\Big]. \quad (5)$$

Here $\overset{a.s.}{\to}$ is almost sure convergence [Rosenthal, 2006, sec. 5.2]. The mean $\boldsymbol{m}$ and covariance $\boldsymbol{K}$ are calculated under the assumption that all the G*-vars are jointly Gaussian, like in section 3.*

Proof sketch. The proof is by induction on the number of G-vars included in the output, added in order of definition. The induction invariant is that, for some $m < M$, eq. 5 holds; and that a subset of G-vars in $[m]$ which form a basis have a non-singular distribution. The detailed proof is in Yang [2019, Appendix H]. □

The following corollary is a consequence of the Master theorem (B.7) and the Central Limit Theorem.

Corollary B.8 (Corollary 5.5, abridged, Yang, 2019)**.** *Fix any* NETSOR *program which satisfies assumptions B.3 to B.6. For simplicity, fix the widths of all the variables to n. The program outputs are $(o_1^\mathsf{T} x_1,\ \dots,\ v_K^\mathsf{T} x_K)$, where each x_k is an* H*-var, and each o_k is a* O*-var. Then, as $n \to \infty$, the output tuple converges in distribution to a Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{K})$. The covariance $\boldsymbol{K}$ is given by doing calculations like section 3, assuming that* G*-vars are jointly Gaussian.*

B.3 NETSOR PROGRAM AND GP BEHAVIOUR: CNN WITH CORRELATED WEIGHTS

NETSOR only has native support for matrix-vector multiplications and linear combinations with constants. How can we represent a convolution operation for convolutional neural network (CNN)? Consider the convolutional layer definition (eq. 1). Changing the sum order, we obtain

Expanding the convolution into a sum, and changing the sum order, we obtain

$$Z_{i,\boldsymbol{q}}^{(\ell)}(\boldsymbol{X}) = \sum_{\boldsymbol{p}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} \sum_{j=1}^{C^{(\ell-1)}} W_{i,j,\boldsymbol{p}}^{(\ell)}\, \boldsymbol{A}_{j,\tilde{\boldsymbol{q}}(\boldsymbol{p})}^{(\ell-1)}(\boldsymbol{X}), \quad (6)$$

which is just a spatial sum of matrix multiplications

$$\mathbf{Z}_{:,\boldsymbol{q}}^{(\ell)}(\boldsymbol{X}) = \sum_{\boldsymbol{p}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} \boldsymbol{W}_{:,:,\boldsymbol{p}}^{(\ell)}\, \boldsymbol{A}_{:,\tilde{\boldsymbol{q}}(\boldsymbol{p})}^{(\ell-1)}(\boldsymbol{X}). \quad (7)$$

Thus, we may express a convolution with multiple filters as a sum of matrix-vector multiplications. This is the canonical way to represent convolutional filters in NETSOR [Yang, 2019, NETSOR program 4].

Here we run into a problem. Equation 9 states that CNN filters are spatially correlated, but assumption B.4 states that A-vars have to be independent. To solve this, we will use the the following well-known lemma, which is the $\boldsymbol{R}\epsilon$ expression of a Gaussian random variable with mean zero. The tensor $\boldsymbol{R}$ is a square root of the covariance.

Lemma B.9. *Let $\boldsymbol{\Sigma} \in \mathbb{R}^{\boldsymbol{P}^2}$ be an arbitrary real-valued covariance tensor. Then there exists another real-valued tensor $\boldsymbol{R} \in \mathbb{R}^{\boldsymbol{P}^2}$ such that $\Sigma_{\boldsymbol{q},\boldsymbol{q}'} = \sum_{\boldsymbol{p}=1}^{\boldsymbol{P}} R_{\boldsymbol{q},\boldsymbol{p}} R_{\boldsymbol{q}',\boldsymbol{p}}$. Next, let $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{R}^{\boldsymbol{P}}$ be real-valued tensors, such that $\boldsymbol{w} = \boldsymbol{R}\boldsymbol{u}$. Suppose the elements of $\boldsymbol{u}$ are i.i.d. standard Gaussian variables, $\{u_{\boldsymbol{p}}\}_{\boldsymbol{p}\in[\boldsymbol{P}]} \overset{iid}{\sim} \mathcal{N}(0,1)$. Then, $\boldsymbol{w}$ has a multivariate Gaussian distribution with mean zero and covariance tensor $\boldsymbol{\Sigma}$.*

Proof. Let $K = |\boldsymbol{P}|$, and $\tilde{\boldsymbol{\Sigma}}$ be $K \times K$ matrices, obtained by flattening the dimensions of $\boldsymbol{\Sigma}$ respectively. Then $\tilde{\boldsymbol{\Sigma}}$ is a real-valued covariance matrix, so it is positive semi-definite and a square matrix $\tilde{\boldsymbol{R}}$ s.t. $\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}^\mathsf{T} = \tilde{\boldsymbol{\Sigma}}$ always exists. Un-flattening $\tilde{\boldsymbol{R}}$ we obtain $\boldsymbol{R}$. The variable $\boldsymbol{w}$ is Gaussian because it is a linear transformation of the Gaussian $\boldsymbol{u}$. Calculating the second moment of $\boldsymbol{w}$ finishes the proof. □

Thus, to express convolution in NETSOR with correlated weights $\boldsymbol{w}$, we can use the following strategy. First, express several convolutions with uncorrelated weights $\boldsymbol{u}$, using eq. 7. Then, combine the output of the convolutions using `LinComb` and coefficients of the tensor $\boldsymbol{R}$.

Given a collection of A-vars $\left\{\boldsymbol{U}_{:,:,\boldsymbol{p}}^{(\ell)}\right\}_{\boldsymbol{p}\in[\boldsymbol{P}^{(\ell)}]}$, we can express the convolutional weights $\boldsymbol{W}^{(\ell)}$ which have covariance $\boldsymbol{\Sigma}^{(\ell)} = \boldsymbol{R}^{(\ell)}\big(\boldsymbol{R}^{(\ell)}\big)^\mathsf{T}$ as

$$\boldsymbol{W}_{:,:,\boldsymbol{p}}^{(\ell)} = \sum_{\boldsymbol{s}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} R_{\boldsymbol{p},\boldsymbol{s}}^{(\ell)} \boldsymbol{U}_{:,:,\boldsymbol{s}}^{(\ell)}\,. \quad (8)$$

Substituting this into eq. 7 we obtain

$$\mathbf{Z}_{:,\boldsymbol{q}}^{(\ell)}(\boldsymbol{X}) = \sum_{\boldsymbol{p}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} \sum_{\boldsymbol{s}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} R_{\boldsymbol{p},\boldsymbol{s}}^{(\ell)} \boldsymbol{U}_{:,:,\boldsymbol{s}}^{(\ell)}\, \boldsymbol{A}_{:,\tilde{\boldsymbol{q}}(\boldsymbol{p})}^{(\ell-1)}(\boldsymbol{X}). \quad (9)$$

To express this computation with NETSOR rules we may write

$$\texttt{MatMul:}\quad \boldsymbol{H}^{(\ell)}_{:,\boldsymbol{s},\boldsymbol{p}}(\boldsymbol{X}) \triangleq \boldsymbol{U}^{(\ell)}_{:,:,\boldsymbol{s}} \boldsymbol{A}^{(\ell-1)}_{:,\boldsymbol{p}}(\boldsymbol{X}) \quad \text{for } \boldsymbol{s} \in [\boldsymbol{P}^{(\ell)}], \boldsymbol{p} \in [\boldsymbol{F}^{(\ell-1)}], \tag{10}$$

$$\texttt{LinComb:}\quad \mathbf{Z}^{(\ell)}_{:,\boldsymbol{q}}(\boldsymbol{X}) \triangleq + \sum_{\boldsymbol{p}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} \sum_{\boldsymbol{s}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} R^{(\ell)}_{\boldsymbol{p},\boldsymbol{s}}\, \boldsymbol{H}^{(\ell)}_{:,\boldsymbol{s},\tilde{\boldsymbol{q}}(\boldsymbol{p})}(\boldsymbol{X}) \quad \text{for } \boldsymbol{q} \in [\boldsymbol{F}^{(\ell)}]. \tag{11}$$

Algorithm 1 uses this construction for every layer to express an L-layer CNN with correlated weights, applied to an input data set $\mathcal{X} \triangleq [\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M]$.

Theorem B.10 (Correlated CNN behaves like a GP)**.** *Consider a countable set of input points $\tilde{\mathcal{X}}$, and a fixed number of layers L. Apply the L-layer convolutional neural network (eqs. 1 and 2) with correlated weights (eq. 9) to $\mathcal{X}$. Assume its nonlinearities are controlled (assumption B.3). For simplicity, fix all layers to have the same number of channels: $C = C^{(1)} = \cdots = C^{(L)}$. Then, as the number of channels $C \to \infty$, the activations $\mathbf{Z}^{(L)}\left(\tilde{\mathcal{X}}\right)$ converge in distribution to a Gaussian process with mean function $\mathbb{E}\left[Z^{(L)}_i\left(\tilde{\mathcal{X}}\right)\right] = m^{(L)}\left(\tilde{\mathcal{X}}\right)$ and covariance function $\mathbb{C}\left[Z^{(L)}_{i,\boldsymbol{q}}\left(\tilde{\mathcal{X}}\right), Z^{(L)}_{i',\boldsymbol{q}'}\left(\tilde{\mathcal{X}}\right)\right] = \delta_{i,i'} K^{(L)}_{\boldsymbol{q},\boldsymbol{q}'}\left(\tilde{\mathcal{X}}, \tilde{\mathcal{X}}\right)$ (section 3).*

Proof. We need to show

1. that algorithm 1, including the postprocessing part, implements a correlated-weight CNN correctly,
2. that the full program converges weakly to a Gaussian process (GP) on $\tilde{\mathcal{X}}$,
3. that the moments of this GP match the ones in section 3.

For the first claim, the key is the equivalence between a convolutional layer with correlated weights (eq. 1), and a spatial outer product followed by linear combination (eqs. 10 and 11). Keeping this in mind, we can verify by inspection that the steps of algorithm 1, including the output postprocessing, implement the recursive CNN equations (eqs. 1 and 2).

The second claim is somewhat more involved. Invoking the Kolmogorov extension theorem [Tao, 2011, Thm. 2.4.3] we restrict our attention to finite subsets $\mathcal{X} \subseteq \tilde{\mathcal{X}}$, which are going to be compatible distributions if claim 3 is true. Since $\boldsymbol{X} \in \mathcal{X}$ are tensors, we may use the Euclidean metric. We then show by theorem B.7 that the output tuple of the CNN NETSOR program converges weakly to a GP as $C \to \infty$. Each activation in the postprocessing is defined as a linear combination of a Gaussian random variable (RV) (the bias) and a RV that converges weakly to a Gaussian, and thus the

Algorithm 1 NETSOR description of an L-layer CNN with correlated weights, with input $\mathcal{X}$.

```
/* G-vars for layer 1 activations, for
   all spatial locations p and input
   points X_m.                                  */
```
Input : $Z^{(1)}_{\boldsymbol{p}}(\boldsymbol{X}_m) : \mathsf{G}(C^{(1)})$ for $\boldsymbol{p} \in [\boldsymbol{F}^{(1)}]$ and $m = 1, \ldots, M$.
```
/* A-vars for the independent
   convolutional weights                        */
```
Input : $\boldsymbol{U}^{(\ell)}_{\boldsymbol{p}} : \mathsf{A}\big(C^{(\ell)}, C^{(\ell-1)}\big)$ for $\boldsymbol{p} \in [\boldsymbol{P}^{(\ell)}]$ and $\ell = 2, \ldots, L-1$.
```
/* O-vars for the output, for every
   patch location s and channel i               */
```
Input : $\boldsymbol{o}_{i,\boldsymbol{s}} : \mathsf{O}\big(C^{(L-1)}\big)$ for $\boldsymbol{s} \in [\boldsymbol{P}^{(L)}]$ and $i = 1, \ldots, C^{(L)}$.

for $m = 1, \ldots, M$ *(data points of m)* **do**
  for $\ell = 2, \ldots, L-1$ *(layer ℓ)* **do**
    for $\boldsymbol{p} = \boldsymbol{1}, \ldots, \boldsymbol{F}^{(\ell-1)}$ **do**
      Nonlin: $\mathsf{H}\big(C^{(\ell-1)}\big)$
      $\boldsymbol{A}^{(\ell-1)}_{:,\boldsymbol{p}}(\boldsymbol{X}_m) \triangleq \phi\Big(\mathbf{Z}^{(\ell-1)}_{:,\boldsymbol{p}}(\boldsymbol{X}_m)\Big)$
      for $\boldsymbol{s} = \boldsymbol{1}, \ldots, \boldsymbol{P}^{(\ell)}$ *(patch location $\boldsymbol{s}$)* **do**
        MatMul: $\mathsf{G}\big(C^{(\ell)}\big)$
        $\boldsymbol{H}^{(\ell)}_{:,\boldsymbol{s},\boldsymbol{p}}(\boldsymbol{X}_m) \triangleq \boldsymbol{U}^{(\ell)}_{\boldsymbol{s}} \boldsymbol{A}^{(\ell-1)}_{\boldsymbol{p}}(\boldsymbol{X}_m)$
      end
    end
    for $\boldsymbol{q} = \boldsymbol{1}, \ldots, \boldsymbol{F}^{(\ell)}$ *(spatial location $\boldsymbol{q}$)* **do**
      LinComb: $\mathsf{G}\big(C^{(\ell)}\big) : \mathbf{Z}^{(\ell)}_{:,\boldsymbol{q}}(\boldsymbol{X}_m)$
      $\triangleq \sum_{\boldsymbol{p}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} \sum_{\boldsymbol{s}=\boldsymbol{1}}^{\boldsymbol{P}^{(\ell)}} R^{(\ell)}_{\boldsymbol{p},\boldsymbol{s}}\, \boldsymbol{H}^{(\ell)}_{:,\boldsymbol{s},\tilde{\boldsymbol{q}}(\boldsymbol{p})}(\boldsymbol{X}_m)$
    end
  end
  for $\boldsymbol{p} \in \big[\boldsymbol{F}^{(L-1)}\big]$ *(spatial location $\boldsymbol{p}$)* **do**
    Nonlin: $\mathsf{H}\big(C^{(L-1)}\big)$
    $\boldsymbol{A}^{(L-1)}_{:,\boldsymbol{p}}(\boldsymbol{X}_m) \triangleq \phi\Big(\mathbf{Z}^{(L-1)}_{:,\boldsymbol{p}}(\boldsymbol{X}_m)\Big)$
  end
end
```
/* One output for every spatial
   location p, patch location s,
   channel i and data point m.                  */
```
Output : $\big(\boldsymbol{o}^{\mathsf{T}}_{i,\boldsymbol{s}} \boldsymbol{A}^{(L-1)}_{:,\boldsymbol{p}}(\boldsymbol{X}_m) :$ for $\boldsymbol{p} \in [\boldsymbol{F}^{(L)}], \boldsymbol{s} \in [\boldsymbol{P}^{(L)}], i \in [C^{(L)}]$ and $m \in [M]\big)$

Output postprocessing: correlate the outputs and add biases (not part of NETSOR)

for $m \in [M]$, $i \in [C^{(L)}]$ *and* $\boldsymbol{q} \in \boldsymbol{F}^{(L)}$ **do**
  $Z^{(L)}_{i,\boldsymbol{q}}(\boldsymbol{X}_m) \triangleq \sum_{\boldsymbol{p}=\boldsymbol{1}}^{\boldsymbol{P}^{(L)}} \sum_{\boldsymbol{s}=\boldsymbol{1}}^{\boldsymbol{P}^{(L)}} R^{(L)}_{\boldsymbol{p},\boldsymbol{s}} \left(\boldsymbol{o}^{\mathsf{T}}_{i,\boldsymbol{s}} \boldsymbol{A}^{(L-1)}_{:,\tilde{\boldsymbol{q}}(\boldsymbol{p})}(\boldsymbol{X}_m)\right)$
end

resulting distribution on $\mathcal{X}$ converges weakly to a Gaussian too.

Finally, we have to show that the postprocessed output has the correct kernel. The output tuple and the activations have mean zero, which is correct. We thus compute the covariance of the output tuple in algorithm 1, for $\boldsymbol{X}, \boldsymbol{X}' \in \tilde{\mathcal{X}}$:

$$\mathbb{C}\left[\boldsymbol{o}_{i,\boldsymbol{s}}^{\mathsf{T}} \boldsymbol{A}_{:,\boldsymbol{p}}^{(L-1)}(\boldsymbol{X}), \boldsymbol{o}_{i',\boldsymbol{s}'}^{\mathsf{T}} \boldsymbol{A}_{:,\boldsymbol{p}'}^{(L-1)}(\boldsymbol{X}')\right] = \delta_{i,i'}\delta_{\boldsymbol{s},\boldsymbol{s}'} V_{\boldsymbol{p},\boldsymbol{p}'}^{(L-1)}(\boldsymbol{X}, \boldsymbol{X}'). \quad (12)$$

The delta functions appear because the O-vars and their elements are all independent. Using this, we can calculate the covariance function of the activations

$$\mathbb{C}\left[Z_{i,\boldsymbol{q}}^{(L)}(\boldsymbol{X}), Z_{i',\boldsymbol{q}'}^{(L)}(\boldsymbol{X}')\right] = \sum_{\boldsymbol{p},\boldsymbol{p}'}^{\boldsymbol{P}^{(L)2}} \sum_{\boldsymbol{s},\boldsymbol{s}'}^{\boldsymbol{P}^{(L)2}} R_{\boldsymbol{p},\boldsymbol{s}}^{(L)} R_{\boldsymbol{p}',\boldsymbol{s}'}^{(L)}\, \mathbb{C}\left[\boldsymbol{o}_{i,\boldsymbol{s}}^{\mathsf{T}} \boldsymbol{A}_{:,\tilde{\boldsymbol{q}}(\boldsymbol{p})}^{(L-1)}(\boldsymbol{X}), \boldsymbol{o}_{i',\boldsymbol{s}'}^{\mathsf{T}} \boldsymbol{A}_{:,\tilde{\boldsymbol{q}}'(\boldsymbol{p}')}^{(L-1)}(\boldsymbol{X}')\right]. \quad (13)$$

Substitute the value of the expectations and eliminate one of the sums due to $\delta_{\boldsymbol{s},\boldsymbol{s}'}$,

$$\mathbb{C}\left[Z_{i,\boldsymbol{q}}^{(L)}(\boldsymbol{X}), Z_{i',\boldsymbol{q}'}^{(L)}(\boldsymbol{X}')\right] = \delta_{i,i'}\Big(\sum_{\boldsymbol{p},\boldsymbol{p}'}^{\boldsymbol{P}^{(L)2}} \sum_{\boldsymbol{s}=\boldsymbol{1}}^{\boldsymbol{P}^{(L)}} R_{\boldsymbol{p},\boldsymbol{s}}^{(L)} R_{\boldsymbol{p}',\boldsymbol{s}}^{(L)} V_{\tilde{\boldsymbol{q}}(\boldsymbol{p}),\tilde{\boldsymbol{q}}'(\boldsymbol{p}')}^{(L-1)}(\boldsymbol{X}, \boldsymbol{X}')\Big). \quad (14)$$

Finally, noting that $\boldsymbol{\Sigma}_{\boldsymbol{p},\boldsymbol{p}'}^{(L)} = \sum_{\boldsymbol{s}=\boldsymbol{1}}^{\boldsymbol{P}^{(L)}} R_{\boldsymbol{p},\boldsymbol{s}}^{(L)} R_{\boldsymbol{p}',\boldsymbol{s}}^{(L)}$ (lemma B.9), and using the definition of $K_{\boldsymbol{q},\boldsymbol{q}'}^{(L)}(\boldsymbol{X}, \boldsymbol{X}')$ (eq. 11), we obtain the claim.

$$= \delta_{i,i'}\Big(\sum_{\boldsymbol{p},\boldsymbol{p}'}^{\boldsymbol{P}^{(L)2}} \boldsymbol{\Sigma}_{\boldsymbol{p},\boldsymbol{p}'}^{(L)} V_{\tilde{\boldsymbol{q}}(\boldsymbol{p}),\tilde{\boldsymbol{q}}'(\boldsymbol{p}')}^{(L-1)}(\boldsymbol{X}, \boldsymbol{X}')\Big)$$
$$= \delta_{i,i'} K_{\boldsymbol{q},\boldsymbol{q}'}^{(L)}(\boldsymbol{X}, \boldsymbol{X}'). \quad (15)$$

□

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `https://www.deeplearningbook.org/`.

Jeffrey S Rosenthal. *A First Look At Rigorous Probability Theory*. World Scientific Publishing Company, 2006.

Terence Tao. *An introduction to measure theory*, volume 126. American Mathematical Society Providence, RI, 2011.

Greg Yang. Wide feedforward or recurrent neural networks of any architecture are Gaussian processes. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*. 2019.