Tensor programs

Part I

Eugene Golikov

March 26, 2020

École Polytechnique Fédérale de Lausanne, Switzerland
Former researcher at DeepPavlov.ai, Moscow Institute of Physics and Technology, Russia

Define:

$$\text{width of a network} = \text{the minimal number of nodes in its hidden representations.}$$

Talk subject: neural nets in the limit of infinite width.

Reason:

1. Sufficiently wide nets $\approx$ infinitely wide nets;
2. Infinitely wide nets are much easier to study theoretically;
3. Infinitely wide nets enjoy a number of cool properties (below).

Given He initialization (standard) and certain parameterization, a fully-connected feedforward network w/o shared weights enjoy the following properties:

1. It converges to a **Gaussian process** at initialization as width $\to \infty$ [Matthews et al., 2018];
2. Its GD training dynamics converges to a **kernel GD with a constant kernel** as width $\to \infty$ [Jacot et al., 2018];
3. The spectrum of its **input-output jacobian** can be computed in the limit of infinite width using a **free independence principle** [Pennington et al., 2017].

Tensor programs series [Yang, 2019, Yang, 2020a, Yang, 2020b] prove these properties for a wide class of models as follows:

1. Introduce a wide class of models called *tensor programs;*
2. Prove a *Master theorem* about their limit behavior;
3. Deduce the properties above from the Master theorem.

Overall talk construction strategy:

1. Take one of the properties discussed above;
2. Illustrate it on a simple model;
3. Introduce a class of tensor programs sufficient to express this property;
4. Prove the corresponding Master theorem;
5. Deduce the property from the Master theorem;
6. Proceed with another property.

Convergence to a Gaussian process

Define a neural network recursively:

$$\underbrace{h^{l+1}(\xi) = W^{l+1}x^{l}(\xi)}_{\text{pre-activations, } \in \mathbb{R}^{n_{l+1}}}, \quad \underbrace{x^{l}(\xi) = \phi(h^{l}(\xi))}_{\text{activations, } \in \mathbb{R}^{n_{l}}}, \quad h^{1}(\xi) = W^{1}\xi, \tag{1}$$

where $W^{l+1} \in \mathbb{R}^{n_{l+1} \times n_l}$.

Proposition
Suppose $W_{ij}^{l+1} \sim \mathcal{N}(0, \sigma_w^2/n_l)$ *iid.*
Then $h^{l+1}(\xi)$ *converges to a Gaussian vector with iid entries as* $n_{1:l} \to \infty$ *sequentially.*

Proof idea: sequentially apply a CLT.

Proposition
Suppose $W_{ij}^{l+1} \sim \mathcal{N}(0, \sigma_w^2/n_l)$ *iid.*
Then $h^{l+1}(\xi)$ *converges to a Gaussian vector with iid entries as* $n_{1:l} \to \infty$ *sequentially.*

Proof sketch.

- **Induction base:** $h^1(\xi) = W^1\xi$ is Gaussian with iid entries;
- **Induction step:** suppose $h^l(\xi)$ converges to a Gaussian with iid entries as $n_{1:l-1} \to \infty$:

$$h_\alpha^{l+1} = \sum_{\beta=1}^{n_l} W_{\alpha\beta}^{l+1}\phi(h_\beta^l) = \frac{\sigma_W}{\sqrt{n_l}} \sum_{\beta=1}^{n_l} (\text{iid RVs with zero mean}) \to \text{Gaussian by CLT} \quad (2)$$

as $n_{1:l} \to \infty$ sequentially. Also, h_α^{l+1} and h_β^{l+1} become uncorrelated ($\Leftrightarrow$ independent) as $n_{1:l} \to \infty$.

$\square$

Moreover, for a set of M inputs $\xi_{1:M}$,

$$\begin{pmatrix} h_\alpha^{l+1}(\xi_1) \\ \dots \\ h_\alpha^{l+1}(\xi_M) \end{pmatrix} = \sum_{\beta=1}^{n_l} W_{\alpha\beta}^{l+1} \begin{pmatrix} \phi(h_\alpha^{l}(\xi_1)) \\ \dots \\ \phi(h_\alpha^{l}(\xi_M)) \end{pmatrix} = \frac{\sigma_W}{\sqrt{n_l}} \sum_{\beta=1}^{n_l} (\text{iid random vectors with zero mean}); \quad (3)$$

it converges to a Gaussian vector as $n_{1:l} \to \infty$ sequentially by CLT.

This implies that

1. $h_\alpha^{l+1}(\cdot)$ converges to a **Gaussian process** as $n_{1:l} \to \infty$;
2. Also, $h_\alpha^{l+1}(\cdot)$ become uncorrelated ($\Leftrightarrow$ independent) for different α.

$$\begin{pmatrix} h_1^l(\xi_1) & h_1^l(\xi_2) & \dots & h_1^l(\xi_M) \\ h_2^l(\xi_1) & h_2^l(\xi_2) & \dots & h_2^l(\xi_M) \\ \dots & \dots & \dots & \dots \\ h_{n_l}^l(\xi_1) & h_{n_l}^l(\xi_2) & \dots & h_{n_l}^l(\xi_M) \end{pmatrix}$$

As $n_{1:l-1} \to \infty$ sequentially,

- Batch dimension — converges to a multivariate Gaussian;
- Neuron dimension — converges to a $\mathbb{R}^{n_l}$ vector with iid Gaussian components.

Theorem ([Matthews et al., 2018])
Given a model of the form

$$h^{l+1}(\xi) = W^{l+1}x^l(\xi), \quad x^l(\xi) = \phi(h^l(\xi)), \quad h^1(\xi) = W^1\xi, \tag{4}$$

where $W^{l+1} \in \mathbb{R}^{n_{l+1}\times n_l}$*, suppose* $W^{l+1}_{ij} \sim \mathcal{N}(0, \sigma_w^2/n_l)$ *iid.*

Then, $\forall l$ *as* $n_{1:l} \to \infty$ *sequentially,*

1. *all components of* $h^{l+1}(\xi)$ *become iid* $\forall \xi$*, and*
2. $\forall \alpha \in [n_{l+1}]$ $\forall M \in \mathbb{N}$ $\forall \xi_{1:M}$ $\{h^{l+1}_\alpha(\xi_1), \ldots, h^{l+1}_\alpha(\xi_M)\}$ *converges weakly to* $\mathcal{N}(0, \Sigma^{l+1})$*, where*
$$\Sigma^{l+1}_{ij} = \sigma_W^2 \mathbb{E}_{z_{1:M}\sim\mathcal{N}(0,\Sigma^l)}\phi(z_i)\phi(z_j), \tag{5}$$
and $\Sigma^1_{ij} = \sigma_W^2 \xi_i^T \xi_j$.

Corollary (informal)
A neural net converges to a GP at initialization as $n_{1:} \to \infty$ sequentially.

Off-topic remark: what happens during training?

1. When quadratic loss is optimized with gradient flow, a neural net remains a GP $\forall t > 0$;
2. The result of training corresponds to the result of GP inference iff only the readout layer is trained.

What is missing in the theorem?

1. Non-sequential limits;
2. Structured weights (as in CNNs);
3. Tied weights (as in RNNs);
4. Batch-norms.

$$\text{A NETSOR program} = (\text{a set of input vars, a sequence of commands}),$$

where variables are of three different **types:**

1. A-vars: matrices with iid Gaussian entries;
2. G-vars: vectors with *asymptotically* iid Gaussian entries;
3. H-vars: images of G-vars by coordinatewise nonlinearities.

Each command generates a new variable from the previous ones using one of the following **ops:**

1. MatMul: $(W : \mathsf{A},\ x : \mathsf{H}) \to Wx : \mathsf{G}$;
2. $\mathrm{LinComb}$: $(\{x_i : \mathsf{G},\ a_i \in \mathbb{R}\}_{i=1}^k) \to \sum_{i=1}^k a_i x_i : \mathsf{G}$;
3. Nonlin: $(\{x_i : \mathsf{G}\}_{i=1}^k,\ \phi : \mathbb{R}^k \to \mathbb{R}) \to \phi(x_{1:k}) : \mathsf{H}$.

Algorithm 1 Example: MLP with two hidden layers

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}
  $h^1 := W^1 x + b^1 : \mathsf{G}(n^1)$ $\{\mathrm{LinComb}\}$
  $x^1 := \phi(h^1) : \mathsf{H}(n^1)$ {layer 1 activation; Nonlin}
  $\tilde{h}^2 := W^2 x^1 : \mathsf{G}(n^2)$ $\{\mathrm{MatMul}\}$
  $h^2 := \tilde{h}^2 + b^2 : \mathsf{G}(n^2)$ {layer 2 preactivation; $\mathrm{LinComb}$}
  $x^2 := \phi(h^2) : \mathsf{H}(n^2)$ {layer 2 activation; Nonlin}
Output: $v^\top x^2 / \sqrt{n^2}$

We can absorb $\mathrm{LinComb} + \mathrm{Nonlin}$ into a single Nonlin: $x^2 = \phi(h^2) = \phi(\tilde{h}^2 + b^2) = \bar{\phi}(h^2, b^2)$.

Algorithm 2 Example: MLP with two hidden layers and a batch-norm

Input: $\{W^1 x_k : \mathsf{G}(n^1)\}_{k=1}^B$ {layer 1 embeddings of inputs in a batch}
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}

$\{h_k^1 := W^1 x_k + b^1 : \mathsf{G}(n^1)\}_{k=1}^B$ {LinComb}
$\{x_k^1 := \tilde{\phi}_k(h_{1:B}^1) : \mathsf{H}(n^1)\}_{k=1}^B$ {BN + activation for layer 1 (see below); Nonlin}
$\{\tilde{h}_k^2 := W^2 x_k^1 : \mathsf{G}(n^2)\}_{k=1}^B$ {MatMul}
$\{h_k^2 := \tilde{h}_k^2 + b^2 : \mathsf{G}(n^2)\}_{k=1}^B$ {layer 2 preactivation; LinComb}
$\{x_k^2 := \tilde{\phi}_k(h_{1:B}^2) : \mathsf{H}(n^2)\}_{k=1}^B$ {BN + activation for layer 2; Nonlin}

Output: $\{v^\top x_k^2 / \sqrt{n^2}\}_{k=1}^B$

Here $\tilde{\phi} : \mathbb{R}^B \to \mathbb{R}^B$ is defined as

$$\tilde{\phi}(h^{1:B}) = \phi\left(\frac{h^{1:B} - \mu(h^{1:B})}{\sigma(h^{1:B})}\right), \quad \mu(h^{1:B}) = \frac{1}{B}\sum_{k=1}^B h^k, \quad \sigma(h^{1:B}) = \sqrt{\frac{1}{B}\sum_{k=1}^B (h^k - \mu(h^{1:B})_k)^2}.$$

Initialization assumption:

1. All hidden dimensions are equal to n;
2. An input variable can be either of type A or of type G;
3. $\forall W$: A we sample $W_{\alpha\beta} \sim \mathcal{N}(0, \sigma_W^2/n)$ iid;
4. $\forall \alpha \in [n]$ we sample $\{x_\alpha : x \text{ is an input G-var}\} \sim \mathcal{N}(\mu^{in}, \Sigma^{in})$.

Our goal: compute the distributions of all G-vars in the program in the limit of $n \to \infty$.

Claim:

1. $\forall$ G-var g, its components become iid as $n \to \infty$;
2. $(g^1_\alpha, \ldots, g^M_\alpha)$ becomes jointly Gaussian with mean and covariance defined recursively as follows[1]:

$$\mu(g) = \begin{cases} \mu^{in}(g) & \text{if } g \text{ is an input G-var;} \\ 0 & \text{if } g = Wy \text{ is introduced by } \mathrm{MatMul}. \end{cases} \tag{6}$$

$$\Sigma(g, \bar{g}) = \begin{cases} \Sigma^{in}(g, \bar{g}) & \text{if } g \text{ and } \bar{g} \text{ are input G-vars;} \\ \sigma_W^2 \mathbb{E}_Z \phi(Z)\bar{\phi}(Z) & \text{if } g = W\phi(Z) \text{ and } \bar{g} = W\bar{\phi}(Z) \text{ introduced by } \mathrm{MatMul}; \\ 0 & \text{else.} \end{cases} \tag{7}$$

Here $Z \sim \mathcal{N}(\mu, \Sigma)$ is a set of all previous G-vars.

[1] we have suppressed the $\mathrm{LinComb}$ op for brevity.

$$\mu(g) = \begin{cases} \mu^{in}(g) & \text{if } g \text{ is an input G-var;} \\ 0 & \text{if } g = Wy \text{ is introduced by } \mathrm{MatMul}. \end{cases} \tag{8}$$

$$\Sigma(g, \bar{g}) = \begin{cases} \Sigma^{in}(g, \bar{g}) & \text{if } g \text{ and } \bar{g} \text{ are input G-vars;} \\ \sigma_W^2 \mathbb{E}_Z \phi(Z)\bar{\phi}(Z) & \text{if } g = W\phi(Z) \text{ and } \bar{g} = W\bar{\phi}(Z) \text{ introduced by } \mathrm{MatMul}; \\ 0 & \text{else.} \end{cases} \tag{9}$$

$$\begin{pmatrix} g_\alpha \\ \bar{g}_\alpha \end{pmatrix} = \underbrace{\sum_\beta W_{\alpha\beta} \begin{pmatrix} \phi_\beta(Z) \\ \bar{\phi}_\beta(Z) \end{pmatrix}}_{\substack{\text{one can directly apply CLT} \\ \text{only if } W \text{ and } Z \text{ are independent}}} \xrightarrow[\text{a "CLT heuristic"}]{} \mathcal{N}\left(\begin{pmatrix} \mu(g) \\ \mu(\bar{g}) \end{pmatrix}, \begin{pmatrix} \Sigma(g,g) & \Sigma(g,\bar{g}) \\ \Sigma(\bar{g},g) & \Sigma(\bar{g},\bar{g}) \end{pmatrix} \right). \tag{10}$$

Definition

We say $\phi : \mathbb{R}^k \to \mathbb{R}$ is controlled if $\exists C, c, \epsilon > 0 : \forall x \in \mathbb{R}^k \; |\phi(x)| \leq e^{C\|x\|_2^{2-\epsilon}+c}$.

Theorem (Netsor Master Theorem, [Yang, 2019])

Let the NETSOR *program satisfy the initialization assumption and let all nonlinearities be controlled. Let $g^{1:M}$ be a set of all G-vars in the program. Then for any controlled $\psi : \mathbb{R}^M \to \mathbb{R}$*

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g_\alpha^1, \ldots, g_\alpha^M) \to \mathbb{E}_{Z\sim\mathcal{N}(\mu,\Sigma)}\psi(Z)$$

a.s. as $n \to \infty$, where $\mu = \{\mu(g^i)\}_{i=1}^M$ and $\Sigma = \{\Sigma(g^i, g^j)\}_{i,j=1}^M$.

$$\begin{pmatrix} g_1^1 & g_1^2 & \ldots & g_1^M \\ g_2^1 & g_2^2 & \ldots & g_2^M \\ \ldots & \ldots & \ldots & \ldots \\ g_n^1 & g_n^2 & \ldots & g_n^M \end{pmatrix}$$

- "Batch" dimension — converges to $\mathcal{N}(\mu, \Sigma)$;
- Neuron dimension — converges to a $\mathbb{R}^n$ vector with iid components $\sim \mathcal{N}(\mu(g^i), \Sigma(g^i, g^i))$.

A NETSOR program

- is able to express the **first forward pass** of a wide class of neural nets (i.e. with shared/structured weights, with BNs etc.);
- reveals its limiting Gaussian process behavior.

Questions:

1. Can we express a **backward pass** as a NETSOR program?
2. What is its limiting behavior?

Why do we need the backward pass?

1. It is necessary to express the whole training process as a Netsor program (later);
2. Computing the initial NTK requires the first backward pass to be computed.

Intermedia: a Neural Tangent Kernel

Let $f(\cdot;\theta)$ be a parametric model. We learn it with **gradient descent:**

$$\dot{\theta}_t = -\mathbb{E}_{\xi,y} \left.\frac{\partial \ell(y,z)}{\partial z}\right|_{z=f(\xi;\theta_t)} \nabla_\theta f(\xi;\theta_t).$$

Left-multiply both sides by $\nabla_\theta^T f(\bar{\xi};\theta_t)$:

$$\dot{f}(\bar{\xi};\theta_t) = \nabla_\theta^T f(\bar{\xi};\theta_t)\dot{\theta}_t = -\mathbb{E}_{\xi,y} \left.\frac{\partial \ell(y,z)}{\partial z}\right|_{z=f(\xi;\theta_t)} \nabla_\theta^T f(\bar{\xi};\theta_t)\nabla_\theta f(\xi;\theta_t).$$

Define a **neural tangent kernel** as $\Theta_t(\xi,\bar{\xi}) = \nabla_\theta^T f(\xi;\theta_t)\nabla_\theta f(\bar{\xi};\theta_t)$.

Then we have a **kernel GD in function space:**

$$\dot{f}(\bar{\xi};\theta_t) = -\mathbb{E}_{\xi,y} \left.\frac{\partial \ell(y,z)}{\partial z}\right|_{z=f(\xi;\theta_t)} \Theta_t(\xi,\bar{\xi}).$$

$$\dot{f}(\bar{\xi};\theta_t) = -\mathbb{E}_{\xi,y} \left.\frac{\partial \ell(y,z)}{\partial z}\right|_{z=f(\xi;\theta_t)} \Theta_t(\xi,\bar{\xi}), \qquad \Theta_t(\xi,\bar{\xi}) = \nabla_\theta^T f(\xi;\theta_t) \nabla_\theta f(\bar{\xi};\theta_t).$$

Instantiate the model as $f(\xi;\theta) = v^T x^L(\xi)$, where

$$x^l(\xi) = \phi(h^l(\xi)), \quad h^l(\xi) = W^l x^{l-1}(\xi) \quad \forall l \in [L], \qquad x^0(\xi) = \xi.$$

Factorize weights as $W^l = \omega^l/\sqrt{n_{l-1}}$, $v^T = \omega^{L+1}/\sqrt{n_L}$ — **NTK parameterization.** Hence $\theta = \{\omega^1, \ldots, \omega^{L+1}\}$. Initialize weights as $\omega^l_{\alpha\beta} \sim \mathcal{N}(0,1)$.

Theorem ([Jacot et al., 2018], informal)
Let ϕ be sufficiently regular. Then, as $n_{1:L} \to \infty$ sequentially,

1. $\Theta_0(\xi,\bar{\xi})$ *converges to a deterministic* $\mathring{\Theta}(\xi,\bar{\xi})$*;*
2. *Moreover,* $\exists T > 0 : \forall t < T \ \Theta_t(\xi,\bar{\xi})$ *converges to the same* $\mathring{\Theta}(\xi,\bar{\xi})$*.*

Consider NTK parameterization: $W^l = \omega_l / \sqrt{n_{l-1}}$ and $v^T = \omega_{L+1}/\sqrt{n_L}$; the model is:

$$f(\xi) = \frac{1}{\sqrt{n_L}}\omega^{L+1}x^L(\xi), \quad x^l(\xi) = \phi(h^l(\xi)), \quad h^l(\xi) = \frac{1}{\sqrt{n_{l-1}}}\omega^l x^{l-1}(\xi) \quad \forall l \in [L], \quad x^0(\xi) = \xi.$$

Its NTK is defined as

$$\Theta(\xi, \bar{\xi}) = \nabla_\theta^T f(\xi;\theta)\nabla_\theta f(\bar{\xi};\theta) = \sum_{l=1}^{L+1} \nabla_{\omega^l}^T f(\xi)\nabla_{\omega^l} f(\bar{\xi}); \tag{11}$$

$$\nabla_{\omega^l} f(\xi) = \frac{1}{\sqrt{n_{l-1}}}\nabla_{h^l} f(\xi) x^{l-1,\top}(\xi). \tag{12}$$

Define $dx^l = \sqrt{n_l}\nabla_{x^l} f$ and $dh^l = \sqrt{n_l}\nabla_{h^l} f$:

$$\nabla_{\omega^l} f(\xi) = \frac{1}{\sqrt{n_{l-1}}}\nabla_{h^l} f(\xi) x^{l-1,\top}(\xi) = \frac{1}{\sqrt{n_{l-1} n_l}} dh^l(\xi) x^{l-1,\top}(\xi); \tag{13}$$

$$\Theta(\xi, \bar{\xi}) = \sum_{l=1}^{L+1} \nabla^T_{\omega^l} f(\xi) \nabla_{\omega^l} f(\bar{\xi}) = \sum_{l=1}^{L+1} \left(\frac{dh^{l,\top} d\bar{h}^l}{n_l}\right)\left(\frac{x^{l-1,\top}\bar{x}^{l-1}}{n_{l-1}}\right). \tag{14}$$

Consider the second multiplier:

$$\frac{x^{l-1,\top}\bar{x}^{l-1}}{n_{l-1}} = \frac{1}{n_{l-1}} \sum_{\alpha=1}^{n_{l-1}} \phi(h^{l-1}_\alpha)\phi(\bar{h}^{l-1}_\alpha) = \frac{1}{n_{l-1}} \sum_{\alpha=1}^{n_{l-1}} \psi(h^{l-1}_\alpha, \bar{h}^{l-1}_\alpha) \quad \text{for } \psi(x, y) = \phi(x)\phi(y);$$

its limit exists and is given by the Master Theorem.

Can we compute the limit of the first multiplier in the same way?

Recall $dx^l = \sqrt{n_l}\nabla_{x^l} f$ and $dh^l = \sqrt{n_l}\nabla_{h^l} f$.

Relations between forward and backward passes:

Forward pass:	Backward pass:	Same, in terms of dx^l and dh^l:
$f(\xi) = \frac{1}{\sqrt{n_L}}\omega^{L+1}x^L(\xi)$	$\nabla_{x^L} f(\xi) = \frac{1}{\sqrt{n_L}}\omega^{L+1,\top}$	$dx^L(\xi) = \omega^{L+1,\top}$
$x^l(\xi) = \phi(h^l(\xi))$	$\nabla_{h^l} f(\xi) = \nabla_{x^l} f(\xi) \odot \phi'(h^l(\xi))$	$dh^l(\xi) = dx^l(\xi) \odot \phi'(h^l(\xi))$
$h^l(\xi) = \frac{1}{\sqrt{n_{l-1}}}\omega^l x^{l-1}(\xi)$	$\nabla_{x^{l-1}} f(\xi) = \frac{1}{\sqrt{n_{l-1}}}\omega^{l,\top}\nabla_{h^l} f(\xi)$	$dx^{l-1}(\xi) = \frac{1}{\sqrt{n_l}}\omega^{l,\top} dh^l(\xi)$

For simplicity, assume $n_1 = \ldots = n_L = n$. Recall $W^l = \omega^l/\sqrt{n}$.

Relations between forward and backward passes:

Forward pass:	Backward pass in terms of dx^l and dh^l:
$x^l(\xi) = \phi(h^l(\xi)) : \mathrm{Nonlin}$	$dh^l(\xi) = dx^l(\xi) \odot \phi'(h^l(\xi)) : \mathrm{Nonlin}$
$h^l(\xi) = W^l x^{l-1}(\xi) : \mathrm{MatMul}$	$dx^{l-1}(\xi) = W^{l,\top} dh^l(\xi) : \mathrm{MatMul?}$

Problems:

1. W and $W^\top$ cannot be both input variables since they are dependent;
2. A NETSOR program does not allow for multiplying by a transposed A-var.

A Netsor program cannot express the backward pass!

$$\text{A } \textsc{Netsor}\top \text{ program} = (\text{a set of input vars}, \text{ a sequence of commands}),$$

where variables are of three different **types:**

1. A-vars: matrices with iid Gaussian entries;
2. G-vars: vectors with *asymptotically* iid Gaussian entries;
3. H-vars: images of G-vars by coordinatewise nonlinearities.

Each command generates a new variable from the previous ones using one of the following **ops:**

1. Trsp: $W : \mathsf{A} \to W^T : \mathsf{A}$;
2. MatMul: $(W : \mathsf{A},\ x : \mathsf{H}) \to Wx : \mathsf{G}$;
3. LinComb: $(\{x_i : \mathsf{G},\ a_i \in \mathbb{R}\}_{i=1}^k) \to \sum_{i=1}^k a_i x_i : \mathsf{G}$;
4. Nonlin: $(\{x_i : \mathsf{G}\}_{i=1}^k,\ \phi : \mathbb{R}^k \to \mathbb{R}) \to \phi(x_{1:k}) : \mathsf{H}$.

Can we keep the same symbolic rules for mean and covariance of G-vars?

$$\mu(g) = \begin{cases} \mu^{in}(g) & \text{if } g \text{ is an input G-var;} \\ 0 & \text{if } g = Wy \text{ is introduced by MatMul.} \end{cases} \tag{15}$$

$$\Sigma(g, \bar{g}) = \begin{cases} \Sigma^{in}(g, \bar{g}) & \text{if } g \text{ and } \bar{g} \text{ are input G-vars;} \\ \sigma_W^2 \mathbb{E}_Z \phi(Z)\bar{\phi}(Z) & \text{if } g = W\phi(Z) \text{ and } \bar{g} = W\bar{\phi}(Z) \text{ introduced by MatMul;} \\ 0 & \text{else.} \end{cases} \tag{16}$$

Here $Z \sim \mathcal{N}(\mu, \Sigma)$ is a set of all previous G-vars.

Consider one of the symbolic rules:

$$\mu(g) = 0 \quad \text{if } g = Wy \text{ is introduced by } \mathrm{MatMul}. \tag{17}$$

Examples:

$$(WWx)_\alpha = \sum_{\beta,\gamma} W_{\alpha\beta} W_{\beta\gamma} x_\gamma = \underbrace{\sum_{\beta\neq\alpha} \left(W_{\alpha\beta} \sum_{\gamma} W_{\beta\gamma} x_\gamma \right)}_{\substack{\text{two sums of iid zero-mean terms;} \\ \text{converges to a zero-mean Gaussian by CLT}}} + \underbrace{W_{\alpha\alpha} \sum_{\gamma\neq\alpha} W_{\alpha\gamma} x_\gamma}_{O(1/\sqrt{n})} + \underbrace{W_{\alpha\alpha}^2 x_\alpha}_{O(1/n)}.$$

$$(W W^T x)_\alpha = \sum_{\beta,\gamma} W_{\alpha\beta} W_{\gamma\beta} x_\gamma = \underbrace{\sum_{\beta} \left(W_{\alpha\beta} \sum_{\gamma\neq\alpha} W_{\gamma\beta} x_\gamma \right)}_{\substack{\text{two sums of iid zero-mean terms;} \\ \text{converges to a zero-mean Gaussian by CLT}}} + \underbrace{\sum_{\beta} W_{\alpha\beta}^2 x_\alpha}_{\text{converges to } \sigma_W^2 \mu(x) \text{ by LLN}}.$$

The previous symbolic rules are not applicable for Netsor$\top$ programs!

The previous symbolic rules are not applicable for **general** NETSOR⊤ programs,

but

they are applicable to NETSOR⊤ programs expressing backpropagation.

Claim: the rule

$$\mu(g) = 0 \quad \text{if } g = Wy \text{ is introduced by } \mathrm{MatMul}. \tag{18}$$

works for NETSOR⊤ programs expressing backpropagation.

Consider $dx^{l-1} = W^{l,T}(dx^l \odot \phi'(h^l))$. Let $\phi(z) = z^2/2$:

$$\begin{aligned} dx^{l-1}_\alpha &= (W^{l,T}(dx^l \odot \phi'(h^l)))_\alpha = (W^{l,T}(dx^l \odot \phi'(W^l x^{l-1})))_\alpha = \\ &= \sum_\beta W^l_{\beta\alpha} dx^l_\beta \phi'(\sum_\gamma W^l_{\beta\gamma} x^{l-1}_\gamma) = \sum_\beta W^l_{\beta\alpha} dx^l_\beta \sum_\gamma W^l_{\beta\gamma} x^{l-1}_\gamma = \\ &= \underbrace{\sum_\beta W^l_{\beta\alpha} dx^l_\beta \sum_{\gamma\neq\beta} W^l_{\beta\gamma} x^{l-1}_\gamma}_{\substack{\text{two sums of iid zero-mean terms;}\\ \text{converges to a zero-mean Gaussian by CLT}}} + \underbrace{x^{l-1}_\alpha \sum_\beta (W^l_{\beta\alpha})^2 dx^l_\beta}_{\text{converges to } x^{l-1}_\alpha \mu(dx^l) \text{ by LLN}} . \end{aligned} \tag{19}$$

Hence $\mu(dx^{l-1}) = \mu(dx^l)$ which by induction implies $\mu(dx^{l-1}) = \mu(dx^L) = \mu(\omega^{L+1}) = 0$.

Proposition
Consider a neural network and a NETSOR⊤ *program expressing its backward pass.*

The symbolic rules for μ *and* Σ *are valid, if*

1. *The output layer has zero mean;*
2. *It is sampled independently from other parameters;*
3. *It is not used anywhere else in the program.*

A general condition that ensures applicability of the previous symbolic rules for μ and Σ:

Condition (BP-likeness)
A $\textsc{Netsor}\top$ *program is said to be BP-like, if there exist input* G*-vars* $v^1, \ldots, v^k$ *such that*

1. *They are sampled with zero mean;*
2. *If* Wz *is used in the program then* z *depends on neither of* $v^1, \ldots, v^k$*;*
3. *If* $W^\top z$ *is used in the program then* z *is an odd function in* $v^1, \ldots, v^k$*.*

Definition
We say $\phi : \mathbb{R}^k \to \mathbb{R}$ is polynomially bounded if $\exists C, c, p > 0 : |\phi(x)| \leq C\|x\|_2^p + c$.

Theorem (Netsor$\top$ Master Theorem, [Yang, 2020a])
Let a $\textsc{Netsor}\top$ *program be BP-like, satisfy the initialization assumption, and let all nonlinearities be polynomially bounded. Let* $g^{1:M}$ *be a set of all G-vars in the program. Then, for any polynomially bounded* $\psi : \mathbb{R}^M \to \mathbb{R}$,

$$\frac{1}{n}\sum_{\alpha=1}^{n} \psi(g_\alpha^1, \ldots, g_\alpha^M) \to \mathbb{E}_{Z\sim\mathcal{N}(\mu,\Sigma)}\psi(Z)$$

a.s. as $n \to \infty$, *where* $\mu = \{\mu(g^i)\}_{i=1}^M$ *and* $\Sigma = \{\Sigma(g^i, g^j)\}_{i,j=1}^M$.

The result is (almost) the same as for Netsor programs!

Back to NTK computation:

$$\Theta(\xi, \bar{\xi}) = \sum_{l=1}^{L+1} \nabla_{\omega^l}^T f(\xi) \nabla_{\omega^l} f(\bar{\xi}) = \sum_{l=1}^{L+1} \left(\frac{dh^{l,\top} d\bar{h}^l}{n} \right) \left(\frac{x^{l-1,\top} \bar{x}^{l-1}}{n} \right). \tag{20}$$

Consider the first multiplier:

$$\frac{dh^{l,\top} \bar{d}h^l}{n} = \frac{1}{n} \sum_{\alpha=1}^{n} dx_\alpha^l d\bar{x}_\alpha^l \phi'(h_\alpha^l) \phi'(\bar{h}_\alpha^l) = \frac{1}{n} \sum_{\alpha=1}^{n} \psi(dx_\alpha^l, d\bar{x}_\alpha^l, h_\alpha^l, \bar{h}_\alpha^l)$$
$$\text{for } \psi(x, y, z, w) = xy\phi'(z)\phi'(w); \tag{21}$$

its limit exists and is given by the Master Theorem.

A BP-like NETSOR⊤ program

- is able to express the **first forward and backward passes** of a wide class of neural nets (i.e. with shared/structured weights, with BNs etc.);
- reveals their limiting Gaussian process behavior;
- can be applied to initial NTK computation.

Questions:

1. Can we express the whole training process as a NETSOR⊤ program?
2. What should be the corresponding Master Theorem?
3. What are the other use-cases of NETSOR⊤ programs?

A teaser for the next part:

1. The random matrices part:
   1.1 Computing an input-ouput jacobian as a **non-BP-like** NETSOR$\top$ program;
   1.2 A general (non-BP-like) NETSOR$\top$ Master Theorem;
   1.3 Free Independence Principle as consequence of the Master Theorem.
2. The learning process part:
   2.1 A learning process as a NETSOR$\top$ program;
   2.2 A maximal update principle.

Jacot, A., Gabriel, F., and Hongler, C. (2018).
Neural tangent kernel: Convergence and generalization in neural networks.
In *Advances in neural information processing systems*, pages 8571–8580.

Matthews, A. G. d. G., Hron, J., Rowland, M., Turner, R. E., and Ghahramani, Z. (2018).
Gaussian process behaviour in wide deep neural networks.
In *International Conference on Learning Representations*.

Pennington, J., Schoenholz, S., and Ganguli, S. (2017).
Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice.
In *Advances in neural information processing systems*, pages 4785–4795.

Yang, G. (2019).
Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes.
arXiv preprint arXiv:1910.12478.

Yang, G. (2020a).

Tensor programs ii: Neural tangent kernel for any architecture.
arXiv preprint arXiv:2006.14548.

Yang, G. (2020b).
Tensor programs iii: Neural matrix laws.
arXiv preprint arXiv:2009.10685.