Understanding and Scaling Large and Deep Neural Networks or "Random Matrix Theory for Extremely Large-Scale ML"

@ Shanghai Jiao Tong University

Zhenyu Liao

based on work of G. Yang at xAI, C. Pehlevan at Harvard, J. Pennington at Google, etc.

School of Electronic Information and Communications
Huazhong University of Science and Technology

October 13, 2024

Outline

1. Motivation: do we (still) need math and theory for modern ML?
2. Math theory for modern ML: a theoretical perspective
3. Math theory for modern ML: a practical perspective

Motivation: do we (still) need math and theory in modern ML?

- **Math has helped a lot in the past**: from Kepler's laws of planetary motion to Newton and calculus
- **AI is doing great**: there is a bit math (in defining problems and computing), but hardly analytic
- for modern AI: **intuition, data, and computation** seem the most important, NOT analytic math theory
- **In this talk**, convey that math theory is still **important** in the design of large-scale ML models, with the example of **Random Matrix Theory (RMT)** for large and deep neural networks (DNNs)

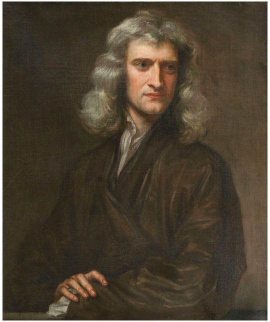


Figure: Portrait of Newton at 46, 1689.

Scaling of sum of independent random variables: LLN and CLT

- **Strong law of large numbers (LLN)**: for a sequence of i.i.d. random variables $x_1, \ldots, x_n$ with the same expectation $\mathbb{E}[x_i] = \mu < \infty$, we have
$$\frac{1}{n}\sum_{i=1}^{n} x_i \to \mu, \tag{1}$$
almost surely as $n \to \infty$.
- **Central limit theorem (CLT)**: for a sequence of i.i.d. random variables $x_1, \ldots, x_n$ with the same expectation $\mathbb{E}[x_i] = \mu$ and variance $\mathrm{Var}[x_i] = \sigma^2 < \infty$, we have
$$\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)\right) \to \mathcal{N}(0, \sigma^2), \tag{2}$$
in distribution as $n \to \infty$.

Consequences of LLN and CLT

For i.i.d. random variables $x_1, \ldots, x_n$ of zero mean and unit variance, e.g., $x_i \sim \mathcal{N}(0,1)$, we have, for n large, the following scaling laws for the sum $\frac{1}{n}\sum_{i=1}^{n} x_i$:

- $\frac{1}{n}\sum_{i=1}^{n} x_i \simeq 0$ by LLN; and
- $\frac{1}{\sqrt{n}}\sum_{i=1}^{n} x_i = O(1)$ with high probability by CLT.

We have known this a bit in the context of DNN

- DNNs involve linear (i.e., weights) and nonlinear (i.e., activation) transformation
- **Xavier initialization** [GB10]: for sigmoid-type activation, randomly initialize a weight matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ having N neurons as

$$[\mathbf{W}]_{ij} \sim \mathcal{N}(0, N^{-1}). \tag{3}$$

```
torch.nn.init.xavier_normal_
```

- **He initialization** [He+15]: for ReLU-type activation, randomly initialize a weight matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ having N neurons as

$$[\mathbf{W}]_{ij} \sim \mathcal{N}(0, 2N^{-1}). \tag{4}$$

```
torch.nn.init.kaiming_normal_
```

- derivation based on **forward propagation**
- similar considerations for CNN, RNN, ResNet, etc.

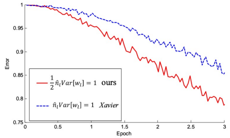


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

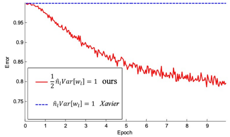


Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But "*Xavier*" (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

Figure: Numerical results in [He+15] for moderately deep NN.

Let us say more on the appropriate scaling of large and deep NNs

Setup and Notations:

- supervised training of an L-layer multi-layer perceptrons (MLP) with full batch gradient flow
- input data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$, denote pre-activation vectors $\mathbf{h}_i^{(\ell)} \in \mathbb{R}^N$ at layer $\ell \in \{1, \ldots, L\}$ as

$$\boxed{\mathbf{h}_i^{(1)} = \frac{1}{N^{a_1}\sqrt{p}} \mathbf{W}^{(1)} \mathbf{x}_i, \quad \mathbf{h}_i^{(\ell)} = \frac{1}{N^{a_\ell}} \mathbf{W}^{(\ell)} \sigma_\ell \left(\mathbf{h}_i^{(\ell-1)} \right)} \quad i \in \{1, \ldots, n\} \tag{5}$$

- scalar output $\boxed{f_{\boldsymbol{\theta}}(\mathbf{x}_i) = \frac{1}{\gamma N^{a_L}} \left(\mathbf{w}^{(L)} \right)^{\mathsf{T}} \sigma_\ell \left(\mathbf{h}_i^{(\ell-1)} \right)}$ for trainable parameters $\boldsymbol{\theta} = \{ \mathbf{W}^{(1)}, \ldots, \mathbf{w}^{(L)} \}$.
- for a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, train the above DNN on the loss function $L(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n L(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)$, with full-batch gradient flow

$$\frac{d\boldsymbol{\theta}}{dt} = -\eta \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \eta \frac{1}{n} \sum_{i=1}^n \Delta_i \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)}{\partial \boldsymbol{\theta}}, \quad \Delta_i \equiv -\frac{\partial L(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)}{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)}, \tag{6}$$

  learning rate $\boxed{\eta = \eta_0 \gamma^2 N^{-c}}$ and **feature learning parameter** $\boxed{\gamma = \gamma_0 N^d}$ for $\eta_0 = \Theta(1)$ and $\gamma_0 = \Theta(1)$
- **initialization scaling scheme**: $\boxed{w_i^{(L)} \sim \mathcal{N}(0, N^{-b_L}), W_{ij}^{(\ell)} \sim \mathcal{N}(0, N^{-b_\ell})}$ and $W_{ij}^{(1)} \sim \mathcal{N}(0, N^{-b_1})$

Appropriate scaling of large and deep NNs

Settings:

- **scaling of NN model**: $\mathbf{h}_i^{(1)} = \frac{1}{N^{a_1}\sqrt{p}} \mathbf{W}^{(1)} \mathbf{x}_i$, $\mathbf{h}_i^{(\ell)} = \frac{1}{N^{a_\ell}} \mathbf{W}^{(\ell)} \sigma_\ell \left(\mathbf{h}_i^{(\ell-1)} \right)$, $f_{\boldsymbol{\theta}}(\mathbf{x}_i) = \frac{1}{\gamma N^{a_L}} \left(\mathbf{w}^{(L)} \right)^\mathsf{T} \sigma_\ell \left(\mathbf{h}_i^{(\ell-1)} \right)$
- **initialization scaling**: $w_i^{(L)} \sim \mathcal{N}(0, N^{-b_L})$, $W_{ij}^{(\ell)} \sim \mathcal{N}(0, N^{-b_\ell})$, and $W_{ij}^{(1)} \sim \mathcal{N}(0, N^{-b_1})$
- **trained under full-batch gradient flow**: $\frac{d\boldsymbol{\theta}}{dt} = -\eta \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \eta \frac{1}{n} \sum_{i=1}^n \Delta_i \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)}{\partial \boldsymbol{\theta}}$ of learning rate $\eta = \eta_0 \gamma^2 N^{-c}$ and feature learning parameter $\gamma = \gamma_0 N^d$ for $\eta_0 = \Theta(1)$ and $\gamma_0 = \Theta(1)$

Objective: for large p, N, achieve **appropriate scaling** on (a, b, c, d) so that

1. **pre-activations $\mathbf{h}^{(\ell)}$ have $\Theta(1)$ entries**:
   - computing the 1st and 2nd moments of $\mathbf{h}^{(1)}$: $\mathbb{E}[\mathbf{h}_i^{(1)}] = \mathbf{0}$, $\mathbb{E}[\mathbf{h}_i^{(1)} (\mathbf{h}_j^{(1)})^\mathsf{T}]_{kq} = \delta_{kq} N^{-(2a_1+b_1)} \cdot \frac{1}{p} \mathbf{x}_i^\mathsf{T} \mathbf{x}_j$; then of $\mathbf{h}^{(\ell)}$
   - we get $\boxed{2a_1 + b_1 = 1}$ and similarly $\boxed{2a_\ell + b_\ell = 1, \ell \in \{1, \ldots, L\}}$
2. **network prediction evolve in $\Theta(1)$ time**:
   - define **feature/conjugate kernel** as the Gram matrix at layer ℓ as $\mathbf{\Phi}^{(\ell)} \in \mathbb{R}^{n \times n}$, $\Phi_{ij}^{(\ell)} = \frac{1}{N} \sigma(\mathbf{h}_i^{(\ell)})^\mathsf{T} \sigma(\mathbf{h}_j^{(\ell)})$
   - under the condition of $\Theta(1)$ pre-activation, it can be shown that in the $N \to \infty$ limit that the pre-activations are **Gaussian process** of zero mean, and covariance given by the (expected) conjugate kernel
   - for $\partial_t f_{\boldsymbol{\theta}}(\cdot) = \Theta(1)$, we get $\boxed{2a_1 + c = 0}$ and $\boxed{2a_\ell + c = 1, \ell \in \{2, \ldots, L\}}$
   - include **classical "mean-field" parameterization** (with $c = 0$, $a_1 = 0$, and $a_\ell = 1/2$) as special case

Appropriate scaling of large and deep NNs

Settings:

- **scaling of NN model**: $\mathbf{h}_i^{(1)} = \frac{1}{N^{a_1}\sqrt{p}} \mathbf{W}^{(1)} \mathbf{x}_i$, $\mathbf{h}_i^{(\ell)} = \frac{1}{N^{a_\ell}} \mathbf{W}^{(\ell)} \sigma_\ell \left(\mathbf{h}_i^{(\ell-1)} \right)$, $f_{\boldsymbol{\theta}}(\mathbf{x}_i) = \frac{1}{\gamma N^{a_L}} \left(\mathbf{w}^{(L)} \right)^\mathsf{T} \sigma_\ell \left(\mathbf{h}_i^{(\ell-1)} \right)$
- **initialization scaling**: $w_i^{(L)} \sim \mathcal{N}(0, N^{-b_L})$, $W_{ij}^{(\ell)} \sim \mathcal{N}(0, N^{-b_\ell})$, and $W_{ij}^{(1)} \sim \mathcal{N}(0, N^{-b_1})$
- **trained under full-batch gradient flow**: $\frac{d\boldsymbol{\theta}}{dt} = -\eta \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \eta \frac{1}{n} \sum_{i=1}^n \Delta_i \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{x}_i)}{\partial \boldsymbol{\theta}}$ of learning rate $\eta = \eta_0 \gamma^2 N^{-c}$ and feature learning parameter $\gamma = \gamma_0 N^d$ for $\eta_0 = \Theta(1)$ and $\gamma_0 = \Theta(1)$

Objective: for large p, N, achieve **appropriate scaling** on (a, b, c, d) so that

3. **features evolve in $\Theta(1)$ time**:
   - by $\partial_t \mathbf{h}_i^{(\ell)} = \Theta(1)$ we have $2a_1 + c - d + 1/2 = 0$, recall that $2a_1 + c = 0$, this is $d = 1/2$, similarly $2a_\ell + c - d - 1/2 = 0$ so that $\boxed{d = 1/2}$
   - in fact, any $d < 1/2$ leads to kernel behavior, and $d = 0$ the **NTK parameterization**

- if further demand raw learning rate $\eta = \Theta(1)$, then parameterization is unique:

$$\boxed{d = 1/2, c = 1, a_\ell = 0, b_\ell = 1, a_1 = -1/2, b_1 = 1} \tag{7}$$

- this is equivalent to the muP parameterization in [YH21]

What is good about this appropriate scaling

- well, things (e.g., DNN pre-activation, evolution of prediction and feature/pre-activation with respect to time) do not scale with the network width N
- BTW, in the case of **ResNet**, a scaling scheme of a similar type can be obtained by considering the infinitely deep $L \to \infty$ limit [Bor+23]
- idea of **maximal update parameterization (muP)** for **hyperparameter transfer** in large models (G. Yang)
- in muP, "narrow" and wide neural networks **share the same set of optimal hyperparameters**, e.g., optimal learning rate (and decay), cross-entropy temperature, initialization scale, regularization, etc.
- one can tune the large model **by just tuning a tiny version** of it and copying over the hyperparameters

Some experiments on muP and μTransfer

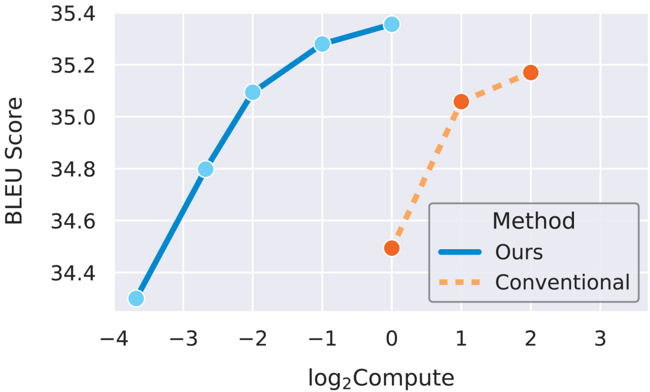


Figure: Comparison μTransfer, which transfers tuned hyperparameters from a small proxy model, with directly tuning the large target model, on IWSLT14 De-En, a machine translation dataset.

Take-away

Take-away messages:

- math/statistics tells a lot about how to scale things, like LLN and CLT
- rather elementary calculus allow to understand the **proper scaling** of large-scale DNN models: for now, **not widely known**
- can be (arguably) applied to transfer optimal hyperparameter design for extremely large-scale models

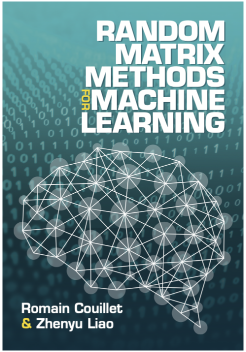
References:

- Tuning GPT-3 on a Single GPU Tensor Programs V, blog by G. Yang. `https://decentdescent.org/tp5.html`
- Cengiz Pehlevan and Blake Bordelon, Lecture Notes on Infinite-Width Limits of Neural Networks, *Princeton Machine Learning Theory Summer School*, 2023.
- Greg Yang and Edward J. Hu. "Tensor Programs IV: Feature Learning in Infinite-Width Neural Networks". In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR, July 2021, pp. 11727–11737

RMT for machine learning: from theory to practice!

Random matrix theory (RMT) for machine learning:

- **change of intuition** from small to large dimensional learning paradigm!
- **better understanding** of existing methods: why they work if they do, and what the issue is if they do not
- **improved novel methods** with performance guarantee!



- book *"Random Matrix Methods for Machine Learning"*
- by Romain Couillet and **Zhenyu Liao**
- Cambridge University Press, 2022
- a pre-production version of the book and exercise solutions at `https://zhenyu-liao.github.io/book/`
- MATLAB and Python codes to reproduce all figures at `https://github.com/Zhenyu-LIAO/RMT4ML`

Thank you! Q & A?