

Maximal Update Parameterization

In the last lecture, we saw that under the NTK parameterization, the intermediate activations only change by $O(m^{-1/2})$. In fact, there is an easy argument for why this is the maximal amount they could move. For simplicity, we assume all hidden layers have width m . Take one step of SGD on an example (x, y) , and hold the input to a hidden layer ℓ fixed. Then

$$\delta h_\ell = -\eta_\ell \ell'(f_\theta(x), y) \|\sigma(h_{\ell-1})\|^2 g_\ell, \quad \delta f_\theta(x) \approx \langle g_\ell, \delta h_\ell \rangle.$$

The update δh_ℓ is proportional to the backward signal g_ℓ , so the two vectors are perfectly aligned and the inner product is as large as it can be. In terms of RMS norms,

$$|\delta f_\theta(x)| \approx \|g_\ell\| \|\delta h_\ell\| = m \|g_\ell\|_{\text{RMS}} \|\delta h_\ell\|_{\text{RMS}}.$$

Keeping $\delta f = O(1)$ therefore forces $\|g_\ell\|_{\text{RMS}} \|\delta h_\ell\|_{\text{RMS}} = O(1/m)$. Under the NTK parameterization $\|g_\ell\|_{\text{RMS}} = \Theta(m^{-1/2})$, so $\|\delta h_\ell\|_{\text{RMS}} = O(m^{-1/2})$, which is exactly what NTK achieves. In other words, **the features are frozen because the output is too sensitive to them.**

This suggests an obvious fix: if we want the features to change by $\Theta(1)$, we need to ensure that the network is less sensitive to them, and more concretely that $\|g_\ell\|_{\text{RMS}} = O(1/m)$.

Shrinking the readout layer

We use the depth L MLP from the last lecture,

$$h_1 = W_1 x, \quad h_\ell = W_\ell \sigma(h_{\ell-1}) \quad (2 \leq \ell \leq L), \quad f_\theta(x) = h_L.$$

How do we scale down g_ℓ by $\sqrt{m}$ without breaking anything else? We still want the hidden activations to have $\Theta(1)$ coordinates at initialization, so fan-in initialization pins down the scale of $W_1, \dots, W_{L-1}$. **The only free parameter left is the output layer W_L .**

For a two layer network $f = a^\top \sigma(Wx)$, we simply have $g = a \odot \sigma'(h)$, so scaling down a by $\sqrt{m}$ scales down g by $\sqrt{m}$ as well. The same is true for deeper networks: the backward pass

$$g_L = 1, \quad g_{L-1} = \sigma'(h_{L-1}) \odot W_L^\top, \quad g_{\ell-1} = \sigma'(h_{\ell-1}) \odot W_\ell^\top g_\ell$$

is *linear* in W_L , and the hidden activations do not depend on W_L at all. So if we initialize $(W_L)_{1j} \sim N(0, 1/m^2)$ instead of $N(0, 1/m)$, every hidden h_ℓ is unchanged. Every g_ℓ shrinks by $\sqrt{m}$, so it now has $\Theta(1/m)$ coordinates, which is what we need for feature learning.

Note the difference from fan-in initialization at the output. If $(W_L)_{1j} \sim N(0, c^2)$, where $c = m^{-1/2}$ for fan-in and $c = m^{-1}$ for μ P, then conditioning on h_{L-1} ,

$$f_\theta(x) = \sum_{j=1}^m (W_L)_{1j} \sigma(h_{L-1})_j \sim N\left(0, c^2 \underbrace{\|\sigma(h_{L-1})\|^2}_{\Theta(m)}\right) = \begin{cases} N(0, \Theta(1)) & \text{fan-in,} \\ N(0, \Theta(1/m)) & \mu\text{P.} \end{cases}$$

So under μ P, $f_\theta(x) \rightarrow 0$ at initialization as $m \rightarrow \infty$.

These are informal course notes and have not undergone peer review. AI tools assisted with simplifying proofs, editing, and typesetting. I take responsibility for the content, and corrections are welcome.

Choosing the learning rates

We can now redo the learning-rate calculation from the last lecture with the new backward pass. Training a hidden layer W_ℓ changes its preactivations by

$$\delta h_\ell = -\eta_{W_\ell} \ell' \underbrace{\|\sigma(h_{\ell-1})\|^2}_{\Theta(m)} \underbrace{g_\ell}_{\Theta(1/m) \text{ coords}},$$

which has $\Theta(1)$ coordinates when $\eta_{W_\ell} = \eta_0$. The input layer is the same except that $\|x\|^2 = \Theta(d)$, so we need $\eta_{W_1} = \eta_0 m/d$. For the output layer, $\|\nabla_{W_L} f\|^2 = \|\sigma(h_{L-1})\|^2 = \Theta(m)$, so we keep $\eta_{W_L} = \eta_0/m$. Then δW_L has entries of order $1/m$, which is the same as the size of W_L itself.

You can check that every layer now contributes $\eta_i \|\nabla_{\theta_i} f\|^2 = \Theta(1)$ to the change in the output, and every layer's activations move by $\Theta(1)$. Each layer is updated as much as possible, which is where the name “maximal update parameterization” (μ P) comes from.

Definition 1 (From NTK to μ P for SGD with LR multipliers, [Yang and Hu, 2021](#)).

	Input W_1	Hidden $W_2, \dots, W_{L-1}$	Output W_L
Initialization scale	$d^{-1/2}$	$m^{-1/2}$	$m^{-1/2} \rightarrow \mathcal{O}(m^{-1})$
Learning rate	$\eta_0/d \rightarrow \eta_0 m/d$	$\eta_0/m \rightarrow \eta_0$	η_0/m

Compared to NTK, the initialization for W_L is scaled down by $\sqrt{m}$ and the learning rate of every layer except W_L is scaled up by m . An easy way to remember the μ P learning rates is that each layer's learning rate scales like **fan-out over fan-in**: m/d for the input layer, $m/m = 1$ for the hidden layers, and $1/m$ for the output layer. This rule assumes that there are no bottlenecks, which we return to below. Here is the updated version of Table 1 from the last lecture:

	Forward h_ℓ	Backward g_ℓ	Update δh_ℓ
RMS norm	$\Theta(1)$	$\Theta(1/m)$	$\Theta(1)$

The general μ P heuristic. The same two steps work for any architecture and any optimizer.

1. **Shrink the readout layer.** The change in f is approximately $\langle g_\ell, \delta h_\ell \rangle$, and these vectors are correlated. Therefore, for feature learning (δh_ℓ has $\Theta(1)$ coordinates), g_ℓ must have coordinates of size $1/m$. This is generally achieved by scaling down the last layer.
2. **Pick the learning rates.** For each layer, choose the learning rate so that one step moves every coordinate of h_ℓ by $\Theta(1)$, including the output $h_L = f$. This is the only step that depends on the optimizer. For SGD it gives $\eta_\ell \propto 1/\|\nabla_{W_\ell} f\|^2$, which is [Definition 1](#). Problem 2 of Homework 1 carries this out for sign-GD, Adam, and Muon.

Warning (aspect ratios and bottlenecks). When the hidden layers have different widths m_ℓ , the NTK limit does not depend on the individual widths, as long as $\min_\ell m_\ell \rightarrow \infty$. This is not true for μ P, and extending μ P to general aspect ratios is nontrivial. For example, one can show that bottleneck layers stifle feature learning. Define the *effective width* of layer j as

its width divided by the squared RMS of its activations at initialization, and the *bottleneck* of layer ℓ as the smaller of its own width and the effective widths after it:

$$\tilde{m}_j := \frac{m_j}{\|h_j^{(0)}\|_{\text{RMS}}^2}, \quad k_\ell := \min\left(m_\ell, \min_{j>\ell} \tilde{m}_j\right).$$

For every ℓ , the maximal update parameterization satisfies

$$\eta_\ell \asymp \eta_0 \frac{k_\ell}{m_{\ell-1}}, \quad \|\delta h_\ell\|_{\text{RMS}} \asymp \sqrt{\frac{k_\ell}{m_\ell}},$$

with $m_0 = d$. Therefore any wide layer before a bottleneck (i.e. $k_\ell \ll m_\ell$) has its features effectively frozen. This also re-derives NTK: the output has width 1 and RMS $\Theta(1)$, so its effective width is 1. Every earlier layer then sees a bottleneck of 1, so its activations can only move by $O(m_\ell^{-1/2})$. In μ P, the output has RMS $\Theta(m^{-1/2})$, so its effective width is m , which removes the bottleneck. Problem 3 of Homework 1 explores this for linear networks.

Practical notes.

- $\eta = \eta_0/m$ says how η scales with width, not how to pick η_0 , which has to be tuned and can differ between layers. For example, if the optimal η is 0.7 at width 1,000, then μ P predicts it should be 0.07 at width 10,000, but it does not predict the 0.7.
- In practice, it is common to initialize $W_L = 0$. The first step then only moves W_L and gives it $\Theta(1/m)$ coordinates, which is the correct scaling for subsequent steps.
- Keep track of every dimension. For example, the MLP in a transformer maps $d \rightarrow 4d \rightarrow d$. Both the input dimension and the width grow with the model size, so treating d as a constant in $\eta = \eta_0 m/d$ gives the wrong answer. Real architectures have many notions of width (embedding dimension, MLP width, number of heads, head dimension, depth, number of experts, ...) and there are μ P rules for each (see [Dey et al., 2025](#)).
- The loss matters too when there are many outputs. For cross-entropy, $\nabla_f \ell = p - e_y$ has ℓ_2 norm $\Theta(1)$ no matter how many classes there are. For the square loss, the gradient scale depends on how the loss is normalized over the classes, and getting this wrong is one reason the square loss was reported to fail with many classes (see [Hui and Belkin, 2021](#)).

We can summarize the three parameterizations we have seen so far, written with learning rate multipliers and fan-in initialization:

	SP	NTK	μ P
Output layer initialization scale	$m^{-1/2}$	$m^{-1/2}$	$O(m^{-1})$
Learning rates (input, hidden, output)	η_0/m for all	$\eta_0/d, \eta_0/m, \eta_0/m$	$\eta_0 m/d, \eta_0, \eta_0/m$
Layers that train	all but the first	all	all
Features move	no	no	yes
Two layer limit	random features	NTK	mean field

Two Layer μ P Gives Mean-Field Dynamics

Consider a two layer network in μ P, written $f = a^\top \sigma(Wx)$ as in Lecture 1, so $W = W_1$ and $a^\top = W_2$. It will be convenient to move the learning rate multipliers into the forward pass, as in the last lecture. We use a common learning rate of $\eta_0 m$. The μ P learning rates are $\eta_0 m/d$ for W and η_0/m for a , so the LR multipliers are $1/d$ and $1/m^2$, and the forward multipliers are their square roots, $1/\sqrt{d}$ and $1/m$. Equivalently, we rescale to $\hat{a} = ma$ and $\hat{W} = \sqrt{d}W$, which both have $\Theta(1)$ entries, and drop the hats. This gives

$$f(x) = \frac{1}{m} \sum_{j=1}^m a_j \sigma\left(\frac{w_j \cdot x}{\sqrt{d}}\right), \quad (a_j, w_j) \stackrel{\text{i.i.d.}}{\sim} \rho_0, \quad \text{learning rate } \eta_0 m.$$

Here ρ_0 is the initialization of a single neuron, for example $N(0, I_{d+1})$, a uniform distribution on a box, or $w \sim N(0, I_d)$ with $a = 0$. For now, we only need that a has mean zero and is independent of w . Since d is fixed, we absorb the $1/\sqrt{d}$ into the input and assume $\|x\| \leq R$. Write $\theta_j = (a_j, w_j) \in \mathbb{R}^{d+1}$ for the j th neuron and $\phi(x; \theta) = a\sigma(w \cdot x)$, so that

$$f(x) = \frac{1}{m} \sum_{j=1}^m \phi(x; \theta_j).$$

This is the same picture as in the first lecture: an infinite-width network is an average $\mathbb{E}_{\theta \sim \rho}[\phi(x; \theta)]$ over a distribution of neurons, and a width m network is a sample of size m from it. The difference is that we will now let the distribution move.

Take a step of SGD on the sample (x_t, y_t) . Each neuron appears in only one term of the average, so $\nabla_{\theta_j} f(x) = \frac{1}{m} \nabla_{\theta} \phi(x; \theta_j)$ and

$$\theta_j \leftarrow \theta_j - \eta_0 m \cdot \frac{1}{m} \ell'(f(x_t), y_t) \nabla_{\theta} \phi(x_t; \theta_j) = \theta_j - \eta_0 \ell'(f(x_t), y_t) \nabla_{\theta} \phi(x_t; \theta_j). \quad (1)$$

The factor of m in the learning rate exactly cancels the $1/m$ in each neuron's gradient. Each neuron moves by $\Theta(1)$ per step, following its own gradient $\nabla_{\theta} \phi$, scaled by how wrong the *whole network* currently is. The neurons only interact through the shared prediction $f(x_t)$.

Note. A nice argument by [Chizat et al. \(2019\)](#) also shows where NTK fits. Put a scale α in front of the average, so that $\alpha = 1$ is mean field and $\alpha = \sqrt{m}$ is the NTK parameterization:

$$f(x) = \frac{\alpha}{m} \sum_{j=1}^m \phi(x; \theta_j).$$

Then by the $\eta \propto 1/\|\nabla_{\theta} f\|^2$ heuristic, the maximal stable learning rate for each neuron scales like $\eta_0 m/\alpha^2$. Therefore, each particle moves by $\Theta(1/\alpha)$ per step. For mean field this recovers the feature learning limit, while for NTK this rederives that each particle moves by $\Theta(1/\sqrt{m})$. In general, any $\alpha \rightarrow \infty$ gives the NTK limit, and the NTK parameterization is the largest α for which the output at initialization, of size $\alpha/\sqrt{m}$, stays bounded.

The infinite-width dynamics

The update (1) depends on the other neurons only through the prediction $f(x_t)$, which is an average over neurons. At initialization the neurons are i.i.d., so $f_0(x)$ concentrates around its expectation

$\bar{f}_0(x) = \mathbb{E}[\phi(x; \theta_0)]$, with fluctuations of order $m^{-1/2}$. Temporarily ignoring these fluctuations, the first step is approximately

$$\theta_j^1 \approx \theta_j^0 - \eta_0 \ell'(\bar{f}_0(x_0), y_0) \nabla_{\theta} \phi(x_0; \theta_j^0),$$

which applies the same deterministic map to every neuron. So to first order, the neurons are still i.i.d. after one step. Therefore, at the next forward pass, f_1 concentrates around its expectation $\bar{f}_1$, again with fluctuations of order $m^{-1/2}$, and to first order the neurons remain i.i.d. after the second step. Repeating this argument, the neurons remain i.i.d. to first order throughout training, and every prediction can be approximated by an expectation. This motivates the following definition, in which we follow a single random neuron and compute the prediction by averaging over its distribution:

Definition 2 (Mean field dynamics, [Mei et al., 2018](#); [Chizat and Bach, 2018](#)). Fix a training sequence (x_t, y_t) . Start from a random neuron $\bar{\theta}_0 \sim \rho_0$, and recursively define

$$\bar{f}_t(x) = \mathbb{E}[\phi(x; \bar{\theta}_t)], \quad \bar{\theta}_{t+1} = \bar{\theta}_t - \eta_0 \ell'(\bar{f}_t(x_t), y_t) \nabla_{\theta} \phi(x_t; \bar{\theta}_t),$$

where the expectation is over $\bar{\theta}_0$. Write ρ_t for the distribution of $\bar{\theta}_t$.

As in Lecture 2, we can compare the finite network against its limit:

	Neural network	Mean field limit
State	m neurons $\theta_1, \dots, \theta_m \in \mathbb{R}^{d+1}$	a distribution ρ_t over $\mathbb{R}^{d+1}$
Initialize	$\theta_j \stackrel{\text{i.i.d.}}{\sim} \rho_0$	ρ_0
Predict	$f_t(x) = \frac{1}{m} \sum_j \phi(x; \theta_j)$	$\bar{f}_t(x) = \mathbb{E}_{\theta \sim \rho_t} [\phi(x; \theta)]$
Update	move each θ_j using $f_t(x_t)$	move every $\theta \sim \rho_t$ using $\bar{f}_t(x_t)$

Unlike the random feature and NTK limits, there is no fixed kernel that summarizes the limit. Its state is an entire distribution over $\mathbb{R}^{d+1}$. On the other hand, the limit is easy to visualize when d is small as a cloud of particles moving in $\mathbb{R}^{d+1}$.

The finite network follows the mean field limit

At initialization, the network is a sample of m i.i.d. neurons from ρ_0 , so Maurey’s lemma from the first lecture gives $\mathbb{E}[(f_0(x) - \bar{f}_0(x))^2] = \text{Var}_{\rho_0}(\phi(x; \theta))/m$, i.e. an error of order $1/\sqrt{m}$. The following theorem says that this remains true after training ([Mei et al., 2018](#); [Chizat and Bach, 2018](#); [Rotskoff and Vanden-Eijnden, 2018](#); [Sirignano and Spiliopoulos, 2020](#)).

Theorem 1 (Mean field limit). *Assume for simplicity that the initialization is bounded, $\|\theta\| \leq K$ for $\theta \sim \rho_0$, σ and ℓ are twice continuously differentiable (e.g. GELU and the square loss), and the data satisfy $\|x_t\|, |y_t| \leq R$. There exists C_T , independent of the width, such that for every $\|x\| \leq R$ and $\delta \in (0, 1)$, with probability at least $1 - \delta$ over the initialization,*

$$|f_T(x) - \bar{f}_T(x)| \leq C_T \sqrt{\frac{\log(2/\delta)}{m}}.$$

Proof. Throughout, $\lesssim$ hides constants that depend on $T, R, K, \eta_0, \sigma, \ell$, but not on m .

For each neuron θ_j^t of the network, we start a *ghost* $\bar{\theta}_j^0 = \theta_j^0$ at the same initialization and move it with the limiting prediction,

$$\bar{\theta}_j^{t+1} = \bar{\theta}_j^t - \eta_0 \bar{\ell}'_t \nabla_{\theta} \phi(x_t; \bar{\theta}_j^t), \quad \bar{\ell}'_t := \ell'(f_t(x_t), y_t).$$

Each ghost follows the dynamics of [Definition 2](#) from its own initialization, and $\bar{f}_t$ is deterministic, so the ghosts are i.i.d. with distribution ρ_t . For any input x , we split the error into a coupling error and a sampling error:

$$\left| f_t(x) - \bar{f}_t(x) \right| \leq \underbrace{\frac{1}{m} \sum_{j=1}^m \left| \phi(x; \theta_j^t) - \phi(x; \bar{\theta}_j^t) \right|}_{\text{coupling error}} + \underbrace{\left| \frac{1}{m} \sum_{j=1}^m \phi(x; \bar{\theta}_j^t) - \bar{f}_t(x) \right|}_{\text{sampling error } \varepsilon_t(x)}. \quad (2)$$

Step 1: localization. We show by induction on t that all neurons and ghosts stay in a ball of radius r_t that does not depend on m . At initialization we can take $r_0 = K$. Now suppose all neurons and ghosts at step t lie in the ball of radius r_t . Then the predictions $f_t(x_t)$ and $\bar{f}_t(x_t)$ are averages of ϕ over this ball, so they are bounded, and the update $\eta_0 \ell'(f, y) \nabla_{\theta} \phi(x; \theta)$ is a continuous function on a compact set, so it is bounded by a constant depending only on r_t . Adding this constant to r_t gives r_{t+1} . Therefore, for $t \leq T$, every neuron and ghost stays in a ball B with radius $\lesssim 1$. Since σ and ℓ are twice continuously differentiable, compactness gives

$$|\phi(x; \theta)| \lesssim 1, \quad \|\nabla_{\theta} \phi(x; \theta)\| \lesssim 1, \quad \|\nabla_{\theta}^2 \phi(x; \theta)\| \lesssim 1 \quad \text{for } \theta \in B,$$

and ℓ' and ℓ'' are bounded on the range of predictions that can occur.

Step 2: the sampling error. Since the ghosts are i.i.d. with distribution ρ_t and bounded by Step 1, for a fixed t and x we can apply Hoeffding's inequality to $\varepsilon_t(x)$. With probability at least $1 - \delta$,

$$\varepsilon_t(x) \lesssim \sqrt{\frac{\log(2/\delta)}{m}}.$$

Let $\varepsilon := \max\{\varepsilon_0(x_0), \dots, \varepsilon_{T-1}(x_{T-1}), \varepsilon_T(x)\}$ be the largest sampling error we will need. Union bounding over these $T + 1$ terms, with probability at least $1 - \delta$,

$$\varepsilon \lesssim \sqrt{\frac{\log(2(T+1)/\delta)}{m}} \lesssim \sqrt{\frac{\log(2/\delta)}{m}},$$

where the last step uses that our constants may depend on T . For the rest of the proof we work on this event, and everything that follows is deterministic.

Step 3: the coupling error. Let

$$D_t := \frac{1}{m} \sum_{j=1}^m \|\theta_j^t - \bar{\theta}_j^t\|$$

be the average distance between the neurons and their ghosts. Since ϕ is Lipschitz on B , the coupling error in (2) is $\lesssim D_t$, so for each of the $T + 1$ pairs (t, x) in the definition of ε ,

$$\left| f_t(x) - \bar{f}_t(x) \right| \lesssim D_t + \varepsilon. \quad (3)$$

To bound D_t , write $\ell'_t := \ell'(f_t(x_t), y_t)$ for the network's loss derivative. Subtracting the two updates and adding and subtracting $\bar{\ell}'_t \nabla_{\theta} \phi(x_t; \theta_j^t)$,

$$\theta_j^{t+1} - \bar{\theta}_j^{t+1} = (\theta_j^t - \bar{\theta}_j^t) - \eta_0 (\ell'_t - \bar{\ell}'_t) \nabla_{\theta} \phi(x_t; \theta_j^t) - \eta_0 \bar{\ell}'_t (\nabla_{\theta} \phi(x_t; \theta_j^t) - \nabla_{\theta} \phi(x_t; \bar{\theta}_j^t)).$$

Taking norms and using the bounds from Step 1,

$$\|\theta_j^{t+1} - \bar{\theta}_j^{t+1}\| \leq \|\theta_j^t - \bar{\theta}_j^t\| + \eta_0 \underbrace{|\ell'_t - \bar{\ell}'_t|}_{\lesssim D_t + \varepsilon} \underbrace{\|\nabla_{\theta}\phi(x_t; \theta_j^t)\|}_{\lesssim 1} + \eta_0 \underbrace{|\bar{\ell}'_t|}_{\lesssim \|\theta_j^t - \bar{\theta}_j^t\|} \underbrace{\|\nabla_{\theta}\phi(x_t; \theta_j^t) - \nabla_{\theta}\phi(x_t; \bar{\theta}_j^t)\|}_{\lesssim \|\theta_j^t - \bar{\theta}_j^t\|},$$

where the first bound uses (3) at $x = x_t$. Averaging over j , there is a constant C such that $D_{t+1} \leq (1 + C)D_t + C\varepsilon$. Since $D_0 = 0$, inducting gives $D_t \leq e^{Ct}\varepsilon \lesssim \varepsilon$. By (3) at $t = T$,

$$|f_T(x) - \bar{f}_T(x)| \lesssim D_T + \varepsilon \lesssim \sqrt{\frac{\log(2/\delta)}{m}}. \quad \square$$

The proof actually shows more: **the neurons are within $\tilde{O}(m^{-1/2})$ of their ghosts, which are i.i.d. samples from ρ_T .** This formalizes the intuition that they remain approximately independent.

Note. This type of result is known as *propagation of chaos* (Sznitman, 1991), where “chaos” is an old word for independence. The key is that the neurons start independent, but they become coupled through the prediction f_t . However, as $m \rightarrow \infty$ each neuron contributes only an infinitesimal amount to f_t , so they regain independence.

It is generally simple to prove propagation of chaos with a time-dependent constant C_T . Significant effort has gone into improving C_T to an absolute constant (uniform-in-time propagation of chaos) or, at the very least, a polynomial dependence on T (Chen et al., 2022; Suzuki et al., 2023; Glasgow et al., 2025), but these analyses tend to be fairly problem specific.

Continuous time

The discrete recursion in Definition 2 is all we need, but the mean field dynamics are usually written in continuous time. Take full-batch gradient descent on a data distribution, let the learning rate go to zero, and measure time s in units of the learning rate. Each neuron then follows the ODE

$$\frac{d\bar{\theta}_s}{ds} = v_{\rho_s}(\bar{\theta}_s), \quad v_{\rho}(\theta) := -\mathbb{E}_{(x,y)}[\ell'(f_{\rho}(x), y) \nabla_{\theta}\phi(x; \theta)], \quad \rho_s = \text{distribution of } \bar{\theta}_s,$$

where $f_{\rho}(x) := \mathbb{E}_{\theta \sim \rho}[\phi(x; \theta)]$ is the prediction of the network with neuron distribution ρ . Because the velocity of each particle depends on the distribution of all the particles, this is an example of a *McKean–Vlasov* equation, and Theorem 1 is an instance of the classical propagation of chaos for such systems (McKean, 1966; Sznitman, 1991; Chainton and Diez, 2022). To see how the distribution evolves, take any test function ψ . By the chain rule,

$$\frac{d}{ds} \mathbb{E}[\psi(\bar{\theta}_s)] = \mathbb{E}[\nabla \psi(\bar{\theta}_s) \cdot v_{\rho_s}(\bar{\theta}_s)].$$

Writing the expectations as integrals and then integrating by parts,

$$\int \psi(\theta) \partial_s \rho_s(\theta) d\theta = \int \nabla \psi(\theta) \cdot v_{\rho_s}(\theta) \rho_s(\theta) d\theta = - \int \psi(\theta) \nabla_{\theta} \cdot (\rho_s v_{\rho_s})(\theta) d\theta.$$

Since this holds for every test function ψ , we get the *continuity equation*

$$\partial_s \rho_s + \nabla_{\theta} \cdot (\rho_s v_{\rho_s}) = 0,$$

which describes the evolution of particles under a changing vector field v_{ρ_s} . This PDE is the form in which the mean field limit usually appears in the literature.

The prediction f_ρ is *linear* in ρ , so if ℓ is convex in its first argument, the population loss

$$L(\rho) := \mathbb{E}_{(x,y)}[\ell(f_\rho(x), y)]$$

is a *convex* function of ρ . However, this does not prevent the mean field dynamics from getting stuck, because they do not perform gradient descent on ρ directly. They can only change ρ by moving the particles continuously. For example, suppose ρ has no mass in some region of parameter space. Even if convexity tells us that adding a few neurons there would decrease the loss, the dynamics cannot create mass there directly. Mass can only flow there continuously from elsewhere.

In fact, the dynamics of ρ are a *Wasserstein gradient flow* of L (Mei et al., 2018; Chizat and Bach, 2018; Rotskoff and Vanden-Eijnden, 2018; Sirignano and Spiliopoulos, 2020). Crucially, L is not convex in this geometry, so the dynamics can potentially get stuck.

Note. Adding noise fixes this problem. If we add Gaussian noise to every particle update, the noise spreads mass over all of parameter space. Slowly annealing the noise to zero then converges to a global minimizer of $L(\rho) + \frac{\lambda}{2} \mathbb{E}_\rho \|\theta\|^2$, where a small weight decay λ keeps particles from escaping to infinity (Chizat, 2022). The price is that in the worst case, the excess loss only decays like $1/\log t$, so it can take exponential time to converge.

Deeper Networks

Why does this particle viewpoint not extend to deeper networks? In a two layer network, $a \in \mathbb{R}^m$ and $W \in \mathbb{R}^{m \times d}$ can be viewed as m particles (a_j, w_j) living in $\mathbb{R}^{d+1}$, a space that does not depend on the width. For a deeper network, each hidden layer $W_\ell \in \mathbb{R}^{m \times m}$ also moves during training, but viewing it as m particles in $\mathbb{R}^m$ does not help, since that space grows with the width. Defining this limit turns out to require significantly more work, and Tensor Programs (Yang and Hu, 2021) and dynamical mean field theory (Bordelon and Pehlevan, 2022) give two equivalent descriptions of it.

References

- B. Bordelon and C. Pehlevan. Self-consistent dynamical field theory of kernel evolution in wide neural networks. In *Advances in Neural Information Processing Systems*, 2022.
- L.-P. Chaintron and A. Diez. Propagation of Chaos: A Review of Models, Methods and Applications. I. Models and Methods. *Kinetic and Related Models*, 15(6), 2022. URL <https://arxiv.org/abs/2203.00446>.
- F. Chen, Z. Ren, and S. Wang. Uniform-in-time Propagation of Chaos for Mean Field Langevin Dynamics, 2022. URL <https://arxiv.org/abs/2212.03050>.
- L. Chizat. Mean-Field Langevin Dynamics: Exponential Convergence and Annealing. *Transactions on Machine Learning Research*, 2022. URL <https://arxiv.org/abs/2202.01009>.
- L. Chizat and F. Bach. On the Global Convergence of Gradient Descent for Over-parameterized Models using Optimal Transport. In *NeurIPS*, 2018. URL <https://arxiv.org/abs/1805.09545>.
- L. Chizat, E. Oyallon, and F. Bach. On Lazy Training in Differentiable Programming. In *NeurIPS*, 2019. URL <https://arxiv.org/abs/1812.07956>.
- N. Dey, B. C. Zhang, L. Noci, M. Li, B. Bordelon, S. Bergsma, C. Pehlevan, B. Hanin, and J. Hestness. Don't be lazy: CompleteP enables compute-efficient deep transformers. In *Advances in Neural Information Processing Systems*, 2025.
- M. Glasgow, D. Wu, and J. Bruna. Propagation of Chaos in One-hidden-layer Neural Networks beyond Logarithmic Time, 2025. URL <https://arxiv.org/abs/2504.13110>.
- L. Hui and M. Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. In *International Conference on Learning Representations*, 2021.
- H. P. McKean. A Class of Markov Processes Associated with Nonlinear Parabolic Equations. *Proceedings of the National Academy of Sciences*, 56(6):1907–1911, 1966.
- S. Mei, A. Montanari, and P.-M. Nguyen. A Mean Field View of the Landscape of Two-Layer Neural Networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018. URL <https://arxiv.org/abs/1804.06561>.
- G. M. Rotskoff and E. Vanden-Eijnden. Parameters as Interacting Particles: Long Time Convergence and Asymptotic Error Scaling of Neural Networks. In *NeurIPS*, 2018. URL <https://arxiv.org/abs/1805.00915>.
- J. Sirignano and K. Spiliopoulos. Mean Field Analysis of Neural Networks: A Law of Large Numbers. *SIAM Journal on Applied Mathematics*, 80(2):725–752, 2020. URL <https://arxiv.org/abs/1805.01053>.
- T. Suzuki, D. Wu, and A. Nitanda. Convergence of Mean-field Langevin Dynamics: Time and Space Discretization, Stochastic Gradient, and Variance Reduction. In *NeurIPS*, 2023. URL <https://arxiv.org/abs/2306.07221>.
- A.-S. Sznitman. Topics in Propagation of Chaos. In *École d'Été de Probabilités de Saint-Flour XIX—1989*, volume 1464 of *Lecture Notes in Mathematics*, pages 165–251. Springer, 1991.
- G. Yang and E. J. Hu. Tensor Programs IV: Feature Learning in Infinite-Width Neural Networks. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 11727–11737. PMLR, 2021. URL <https://proceedings.mlr.press/v139/yang21c.html>.