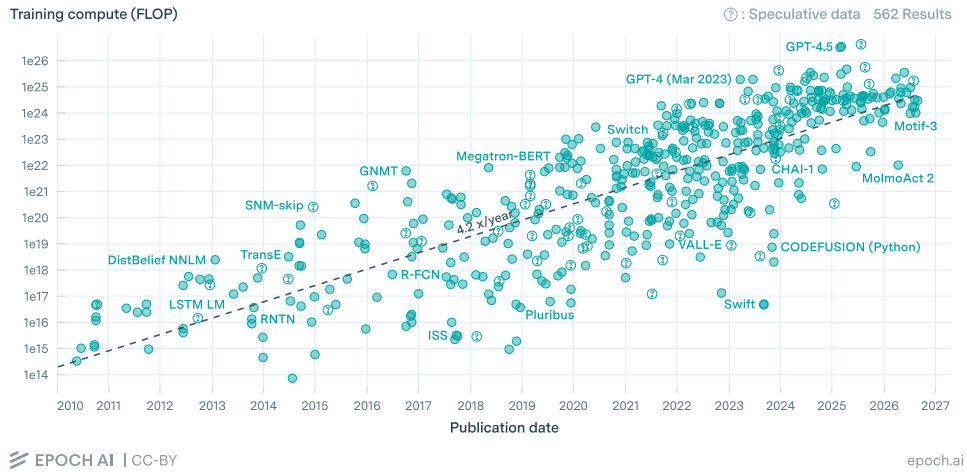


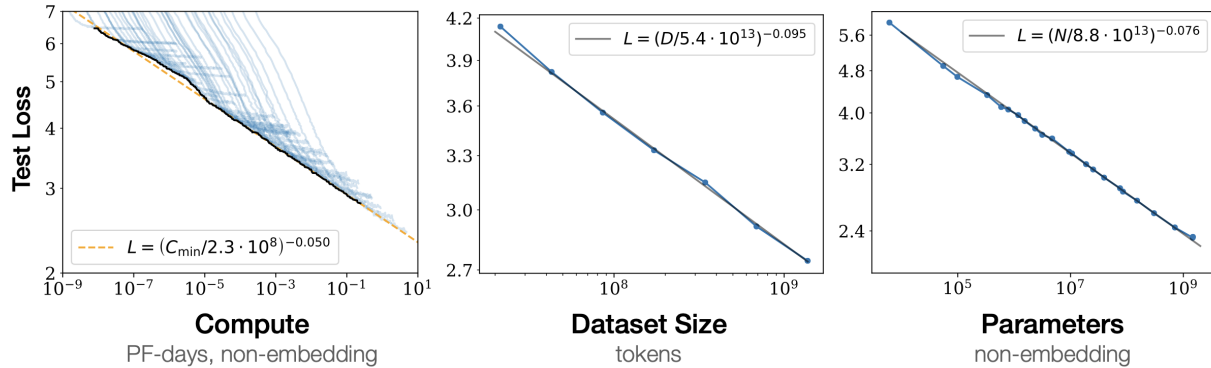
Scaling Limits

Scaling models and training data has been a major driver of progress in deep learning. Training compute per model, roughly $\#param \times \#data$, has doubled about every six months (Epoch AI, 2026):

Notable AI models



This explosion in compute is justified by scaling laws: the empirical observation that larger models trained on more data predictably produce better models, making compute a safe investment:



Language-model test loss as a function of compute, dataset size, and parameter count (Kaplan et al., 2020).

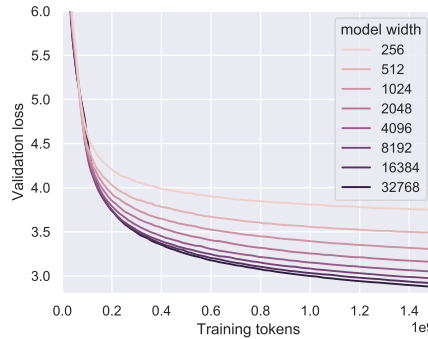
We have a poor understanding of why scaling laws hold, in part because they depend on natural data, and it is hard to say anything meaningful about the distribution of text on the internet. We will therefore abstract away the dataset and ask the simpler question:

What happens when we make the model bigger?

The natural mathematical version of this question is whether, as the model grows, its training dynamics approach those of some idealized “infinite size” network.

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.

The answer is that *it depends*. Specifically, it depends on how you scale the initialization and the optimizer’s learning rates as you scale the model. Different choices give qualitatively different limits: some diverge, some reduce to kernel methods with no feature learning, and others preserve nontrivial feature learning even at infinite width. The focus of this section is characterizing which choices lead to which infinite-size limits, and using that understanding to guide how we scale models.



Loss curves under μP appear to converge to an infinite-width limit. From [Yang et al. \(2022\)](#).

Note. The regimes that this theory can handle are overly idealized, as they focus on scaling the model size while holding the dataset fixed, whereas in practice these grow together. However, it turns out that the scaling strategies suggested by this theory, most notably μP and its descendants, work remarkably well in practice, and this is one of the few lines of work where deep learning theory has provided prescriptions rather than explanations.

Standard Parameterization

Standard parameterization (SP) is the most commonly used initialization scheme, and is the default in deep learning libraries like JAX or PyTorch. The goal of standard parameterization is to initialize a network randomly in a way that maintains $O(1)$ coordinates for all of the intermediate activations throughout the network at initialization. The natural way to measure $O(1)$ coordinates is through the root-mean-square (RMS) norm, which is just a rescaled Euclidean norm. If $h \in \mathbb{R}^d$,

$$\|h\|_{\text{RMS}} := \sqrt{\frac{1}{d} \sum_{i=1}^d h_i^2} = \frac{\|h\|_2}{\sqrt{d}}.$$

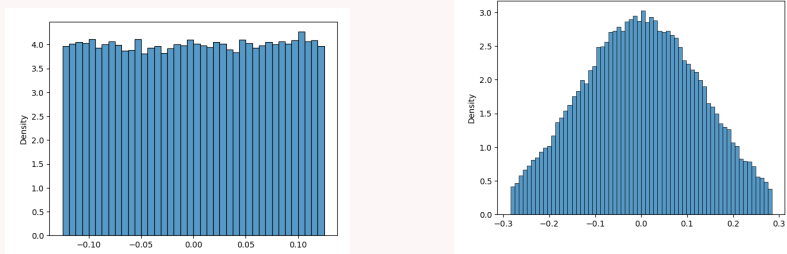
This measures the “average size” of a coordinate. For example, if $h_i \stackrel{i.i.d.}{\sim} N(0, 1)$ then $\|h\|_{\text{RMS}} \approx 1$ for large d . We initialize each layer to approximately preserve the RMS norm of its input. Let $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and initialize $W_{ij} \stackrel{i.i.d.}{\sim} N(0, s^2)$. For a fixed input h ,

$$(Wh)_i \stackrel{i.i.d.}{\sim} N(0, s^2 \|h\|_2^2) = N(0, s^2 d_{\text{in}} \|h\|_{\text{RMS}}^2).$$

This has RMS norm approximately $s\sqrt{d_{\text{in}}}\|h\|_{\text{RMS}}$. Choosing $s = d_{\text{in}}^{-1/2}$ preserves this scale. This is known as *fan-in* or *LeCun* initialization ([LeCun et al., 1998](#)).

Note. A common failure mode when chaining MLP layers is that the activation σ may not preserve the RMS norm. For example, if $h_i \sim N(0, 1)$, $\|\text{ReLU}(h)\|_{\text{RMS}}^2 \approx 1/2$. After L such layers, $\|h_L\|_{\text{RMS}}^2 \approx 2^{-L}$. This can be solved by rescaling $\sigma \rightarrow \sqrt{2} \cdot \sigma$ or by scaling up the initial variance of W by a factor of 2. The latter is known as Kaiming initialization (He et al., 2015).

Warning. Do you know how your framework initializes its linear layers? Flax uses LeCun initialization with variance $1/d_{\text{in}}$, implemented with a truncated normal (Flax, n.d.). However, PyTorch uses $\text{Unif}(-d_{\text{in}}^{-1/2}, d_{\text{in}}^{-1/2})$, which has variance $1/(3d_{\text{in}})$ (PyTorch, n.d.). Even without nonlinearities, this shrinks the RMS scale by $1/\sqrt{3}$ per layer, giving $3^{-L/2}$ over L layers.



PyTorch's uniform initialization (left) and Flax's LeCun initialization (right).

Optimization in SP

Experimental setup. The MNIST experiments below classify handwritten 3s versus 5s using 1,024 examples, a two-layer network with standard parameterization, MSE loss, and 500 full-batch gradient-descent updates. Widths range from 8 to 32,768. The sweeps use three seeds.



Once we've initialized the network, what learning rate η should we use?

To start, let us decompose the loss into the model f and the loss criterion ℓ :

$$L(\theta) = \ell(f_{\theta}(x), y).$$

For example, MSE is $\ell(f, y) = \frac{1}{2}(f - y)^2$ and logistic loss is $\ell(f, y) = \log(1 + e^{-fy})$. We can then decompose the gradient of the loss using the chain rule:

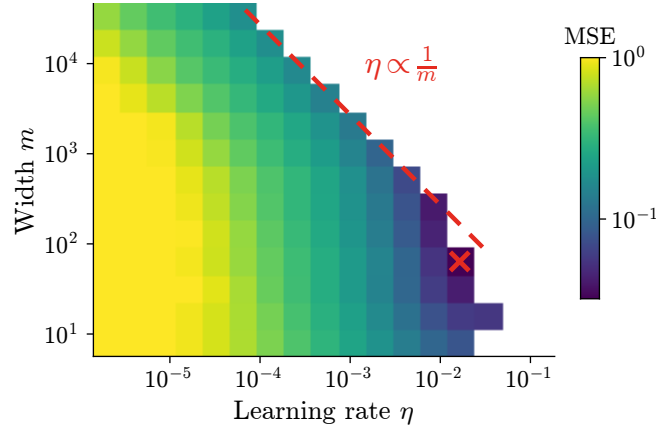
$$\nabla L(\theta) = \ell'(f, y) \nabla_{\theta} f_{\theta}(x).$$

A gradient step $\Delta\theta = -\eta \nabla L(\theta)$ therefore changes the prediction, to first order, by

$$\delta f_{\theta}(x) \approx \langle \nabla_{\theta} f_{\theta}(x), \Delta\theta \rangle = -\eta \ell'(f, y) \|\nabla_{\theta} f_{\theta}(x)\|^2.$$

When $\ell'(f, y) = O(1)$, keeping the prediction update order one suggests

$$\eta \asymp \frac{1}{\|\nabla_{\theta} f_{\theta}(x)\|^2}.$$



Mean training loss over the last 50 of 500 updates. White cells indicate divergence. The stable learning-rate range shrinks as $1/m$. The best tested pair, $(m, \eta) \approx (64, 0.016)$, is denoted by the red $\times$.

Note. Splitting θ into parameter groups $(\theta_1, \dots, \theta_L)$ gives

$$\delta f_\theta(x) \approx -\eta \ell'(f, y) \sum_{i=1}^L \underbrace{\|\nabla_{\theta_i} f_\theta(x)\|^2}_{\text{contribution of group } i}.$$

This lets us read off which parameter groups affect the dynamics as width grows.

We can now instantiate this at initialization for a two layer neural network:

$$f_\theta(x) = a^T \sigma(Wx).$$

We can compute the gradients:

$$\nabla_a f_\theta(x) = \sigma(Wx), \quad \nabla_W f_\theta(x) = [a \odot \sigma'(Wx)]x^T.$$

For inputs with RMS norm of order one, $\|x\|^2 = \Theta(d)$. Under SP, the activations have order-one coordinates and a has coordinates of size $m^{-1/2}$, so the typical squared gradient norms at initialization are:

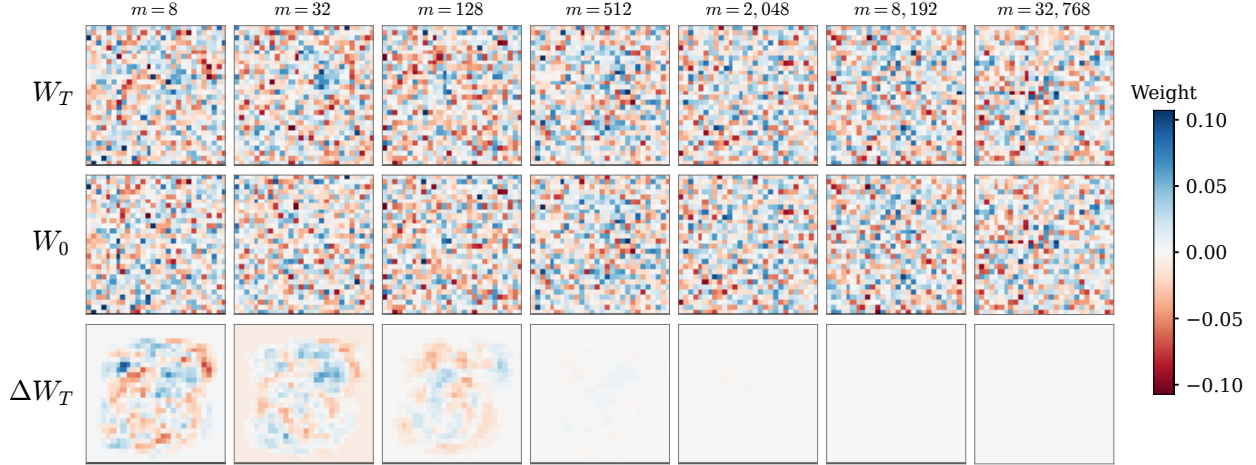
$$\begin{aligned} \|\nabla_a f_\theta(x)\|^2 &= \sum_{j \in [m]} \sigma(w_j \cdot x)^2 = \Theta(m) \\ \|\nabla_W f_\theta(x)\|_F^2 &= \|x\|^2 \sum_{j \in [m]} a_j^2 \sigma'(w_j \cdot x)^2 = \Theta(d) \end{aligned}$$

Therefore, if $m \gg d$, we need to set $\eta \lesssim 1/m$ to avoid the predictions blowing up after a single step.

However, this also shows something quite striking. If we set $\eta = 1/m$,

$$\delta f_\theta(x) \approx -\ell'(f, y) \left[\underbrace{\eta \|\nabla_W f_\theta(x)\|_F^2}_{\Theta(d/m)} + \underbrace{\eta \|\nabla_a f_\theta(x)\|^2}_{\Theta(1)} \right].$$

This suggests that the first layer has a vanishing effect on the predictions as $m \rightarrow \infty$ with d fixed.



Representative neurons after training, decomposed as $w_T = w_0 + \Delta w_T$. Each $w_t \in \mathbb{R}^{784}$ is reshaped as a 28×28 image for visualization. As the width grows, the difference Δw_T shrinks to zero.

To justify this over multiple steps, we compare training both layers with training only the readout. First, let $\theta_t = (a_t, W_t)$ denote the network parameters after t SGD steps with $\eta = \eta_0/m$ on the sequence of samples (x_t, y_t) :

$$a_{t+1} = a_t - \eta \ell'_t \sigma(W_t x_t), \quad W_{t+1} = W_t - \eta \ell'_t [a_t \odot \sigma'(W_t x_t)] x_t^\top, \quad \ell'_t := \ell'(f_{\theta_t}(x_t), y_t).$$

We similarly define the **random features model** $\tilde{\theta}_t = (\tilde{a}_t, W_0)$, which uses the same initialization, sample sequence, and learning rate but trains only $\tilde{a}_t$.

Under SP, the initialization satisfies $\|a_0\| = O(1)$ and $\|W_0\|_2 = O(\sqrt{m})$ with high probability. Conditioned on this high probability event, we can prove that the neural network and the random features model have equivalent dynamics as $m \rightarrow \infty$:

Theorem 1 (SP freezes the first layer). *Assume σ is Lipschitz, all training inputs satisfy $\|x_t\| \leq R$, and ℓ', ℓ'' are bounded.¹ Suppose $\|a_0\| = O(1)$ and $\|W_0\|_2 = O(\sqrt{m})$. For every fixed number of steps T , there exists C_T , independent of the width, such that*

$$\sup_{\|x\| \leq R} |f_{\theta_T}(x) - f_{\tilde{\theta}_T}(x)| \leq \frac{C_T}{m}.$$

Proof. Write $\phi_t(x) = \sigma(W_t x)$, and abbreviate the two predictions as $f_t(x)$ and $\tilde{f}_t(x)$. All bounds below are for fixed T . The notation $\lesssim$ hides constants independent of width.

First, we show that $\|a_t\| = O(1)$ and $\|W_t\|_2 = O(\sqrt{m})$ persist over finitely many steps. They hold at initialization. Assuming them at step t , Lipschitz continuity gives

$$\|\phi_t(x)\| \lesssim \sqrt{m} |\sigma(0)| + \|W_t\|_2 \|x\| = O(\sqrt{m}).$$

Each step moves a by $\|a_{t+1} - a_t\| = \eta \ell'_t \|\phi_t(x_t)\| = O(m^{-1/2})$ and W by $\|W_{t+1} - W_t\|_F \lesssim \eta \|a_t\| = O(m^{-1})$, preserving both bounds at step $t+1$. Summing also gives $\|W_t - W_0\|_F = O(m^{-1})$. For any $\|x\| \leq R$, Lipschitz continuity of σ then gives

$$\|\phi_t(x) - \phi_0(x)\| = \|\sigma(W_t x) - \sigma(W_0 x)\| \lesssim \|W_t - W_0\|_2 \lesssim \frac{1}{m}.$$

¹This theorem can be substantially generalized at the cost of a more complicated proof.

Here $\|\cdot\|_2$ for a matrix is its operator norm, which is at most its Frobenius norm.

Let $d_t := a_t - \tilde{a}_t$. We will prove by induction that $\|d_t\| = O(m^{-3/2})$. The claim holds initially since $d_0 = 0$. Assuming it at step t , split the prediction error into feature movement and readout error:

$$\begin{aligned} |f_t(x) - \tilde{f}_t(x)| &= |a_t^\top [\phi_t(x) - \phi_0(x)] + d_t^\top \phi_0(x)| \\ &\leq |a_t^\top [\phi_t(x) - \phi_0(x)]| + |d_t^\top \phi_0(x)| \\ &\leq \underbrace{\|a_t\| \|\phi_t(x) - \phi_0(x)\|}_{O(m^{-1})} + \underbrace{\|d_t\| \|\phi_0(x)\|}_{O(m^{-1})}. \end{aligned} \quad (1)$$

Since ℓ' is Lipschitz, $|\ell'_t - \tilde{\ell}'_t| \lesssim |f_t(x_t) - \tilde{f}_t(x_t)| = O(m^{-1})$. Subtract the readout updates, adding and subtracting $\eta \ell'_t \phi_0(x_t)$. The triangle inequality gives

$$\|d_{t+1}\| \leq \underbrace{\|d_t\|}_{O(m^{-3/2})} + \underbrace{\eta |\ell'_t - \tilde{\ell}'_t| \|\phi_0(x_t)\|}_{O(m^{-3/2})} + \underbrace{\eta |\ell'_t| \|\phi_t(x_t) - \phi_0(x_t)\|}_{O(m^{-2})} = O(m^{-3/2}).$$

This completes the induction, and plugging into (1) completes the theorem. $\square$

The Infinite Width Limit

The random feature model $f_t(x) = a_t^\top \phi(x)$ is still a finite-width object: its features $\phi(x) = \sigma(Wx) \in \mathbb{R}^m$ depend on m . Our goal is to define the corresponding infinite-width object. The key idea is to trace the dynamics without evaluating $\phi(x)$. Instead, we keep each feature vector symbolic until it is contracted with another feature vector $\phi(x')$, then read out the result using the kernel $K^{(m)}$:

$$K^{(m)}(x, x') := \frac{1}{m} \langle \phi(x), \phi(x') \rangle. \quad (2)$$

For example, we can use this to trace a step of SGD. Take a step on the sample (x_t, y_t) :

$$a_{t+1} = a_t - \eta \ell'_t \phi(x_t), \quad \ell'_t := \ell'(f_t(x_t), y_t).$$

The prediction at any input x changes by

$$f_{t+1}(x) - f_t(x) = (a_{t+1} - a_t)^\top \phi(x) = -\eta \ell'_t \langle \phi(x_t), \phi(x) \rangle = -\eta_0 \ell'_t K^{(m)}(x, x_t).$$

Thus, the kernel alone gives us enough information to track the predictions made by the random feature model. To chain together multiple steps, let the dataset be $x_1, \dots, x_n$, and write $K_{ij}^{(m)} = K^{(m)}(x_i, x_j)$ and $f = (f(x_1), \dots, f(x_n))^\top$. An SGD step on example i updates the predictions f by

$$f \leftarrow f - \eta_0 \ell'(f_i, y_i) K_{:,i}^{(m)}. \quad (3)$$

We can therefore train by just tracking the n predictions f and the $n \times n$ kernel matrix $K^{(m)}$.

It remains to initialize the predictions. Let Φ have rows $\phi(x_i)^\top$, and recall that $a_0 \sim N(0, I_m/m)$. Conditional on W , the vector $f_0 = \Phi a_0$ is a linear transformation of a Gaussian, so

$$f_0 \mid W \sim N\left(0, \frac{1}{m} \Phi \Phi^\top\right) = N(0, K^{(m)}).$$

Thus $K^{(m)}$ determines both the distribution of the initial predictions and their training dynamics.

This gives us an equivalent way to match the exact training dynamics at finite width: sample $W_{ij} \sim N(0, 1/d)$, compute $K^{(m)}$ using (2), initialize $f \sim N(0, K^{(m)})$, then use (3) to update f .

This kernel viewpoint makes it easy to take the infinite-width limit: we track only n predictions, and all dependence on m is captured by $K^{(m)}$. Taking the limit therefore reduces to finding the limit of $K^{(m)}(x, x')$ as $m \rightarrow \infty$. Since each kernel entry is an average over independent neurons, the law of large numbers implies that the kernel $K^{(m)}$ converges to a limiting kernel $K^{(\infty)}$ as $m \rightarrow \infty$:

$$K^{(m)}(x, x') = \frac{1}{m} \sum_{j=1}^m \sigma(w_j \cdot x) \sigma(w_j \cdot x') \xrightarrow{m \rightarrow \infty} \underbrace{\mathbb{E}_w [\sigma(w \cdot x) \sigma(w \cdot x')]}_{K^{(\infty)}(x, x')} \quad \text{where } w \sim N(0, \tfrac{1}{d} I_d).$$

The limiting kernel is independent of width² and can be computed in closed form for some activations:

Proposition 1 (Arc-cosine kernels, [Cho and Saul, 2009](#)). *For nonzero x, x' , let $\theta = \arccos \frac{x \cdot x'}{\|x\| \|x'\|}$ be the angle between them. With $w \sim N(0, I_d/d)$,*

$$K^{(\infty)}(x, x') = \begin{cases} \frac{\pi - \theta}{2\pi}, & \sigma(z) = \mathbf{1}_{z \geq 0}, \\ \frac{\|x\|_{\text{RMS}} \|x'\|_{\text{RMS}}}{2\pi} [\sin \theta + (\pi - \theta) \cos \theta], & \sigma(z) = \text{ReLU}(z) = \max\{0, z\}. \end{cases}$$

Corollary 1 (The infinite-width dynamics of SP). *Fix a dataset $(x_i, y_i)_{i=1}^n$, a sample sequence (i_t) , and a number of steps T . Under the assumptions of Theorem 1, with Gaussian SP initialization and $\eta = \eta_0/m$, the network's prediction trajectory converges in distribution as $m \rightarrow \infty$ to $(F_0, \dots, F_T)$, where*

$$F_0 \sim N(0, K^{(\infty)}), \quad F_{t+1} = F_t - \eta_0 \ell'((F_t)_{i_t}, y_{i_t}) K_{:,i_t}^{(\infty)}.$$

Note. The limiting object is given by the random feature kernel only for two-layer networks. For deeper networks, SP still freezes the first layer in the limit, but the remaining layers all contribute to the limiting dynamics. The limiting kernel is therefore best described by the Neural Tangent Kernel from the next lecture, but with the first layer frozen.

The following table compares the process of training the network from scratch against directly computing the random feature limit.

	Neural network	Random feature limit
State	$\theta_t = (a_t, W_t) \in \mathbb{R}^m \times \mathbb{R}^{m \times d}$	$f_t \in \mathbb{R}^n$
Initialize features	Sample $(W_0)_{ij} \sim N(0, 1/d)$	Compute $K_{ij}^{(\infty)} = K^{(\infty)}(x_i, x_j)$
Initialize readout	Sample $(a_0)_j \sim N(0, 1/m)$	Sample $f_0 \sim N(0, K^{(\infty)})$
Update	$\theta_{t+1} = \theta_t - \frac{\eta_0}{m} \ell'_t \nabla_{\theta} f_{\theta_t}(x_{i_t})$	$f_{t+1} = f_t - \eta_0 \ell'_t K_{:,i_t}^{(\infty)}$

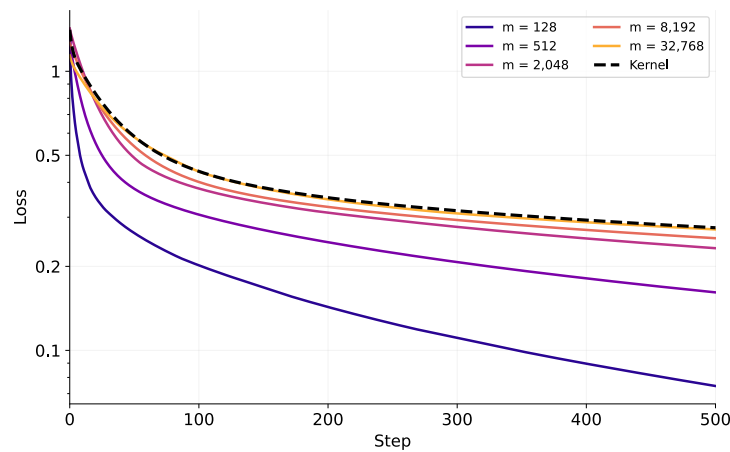
²The random feature kernel here is also known as the neural network Gaussian process (NNGP) kernel ([Neal, 1996](#); [Lee et al., 2018](#)).

We can also compare the computational cost of training the network against directly simulating the limit if we track T SGD steps on n samples:

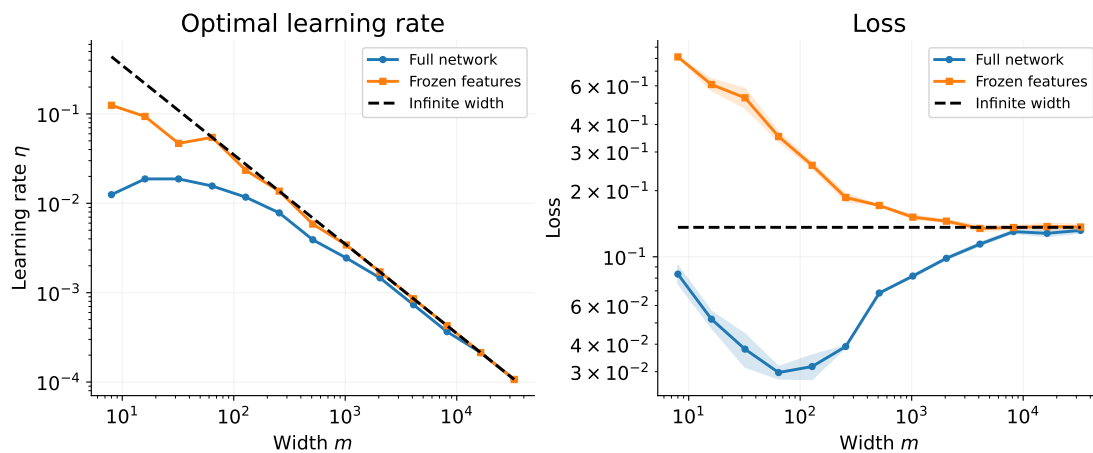
Model	Precompute	Per SGD step	Total
Full network	—	$O(md)$	$O(mdT)$
Random features	$O(mdn)$	$O(m)$	$O(mdn + mT)$
Infinite-width kernel	$O(n^2d)$	$O(n)$	$O(n^2d + nT)$

For one pass over the data, $T = n$, the network and random features both cost $O(mdn)$ while the kernel costs $O(n^2d)$, so the exact limit is cheaper to simulate than the network whenever $m \gg n$.

Empirically verifying the infinite width limit



Training loss trajectories with fixed η_0 and $\eta = \eta_0/m$ approach the dashed line, which represents the limiting dynamics for the random feature kernel. Counterintuitively, wider networks train more slowly than narrower ones in this experiment, which can be attributed to the vanishing contribution of the first layer in the limit.



Left: optimal learning rate as a function of width. Right: best loss as a function of width after tuning the learning rate. Additional width helps until $m = 64$, after which additional width hurts performance, because it drives the dynamics towards the random feature limit.

References

- Y. Cho and L. K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, volume 22, 2009. URL <https://proceedings.neurips.cc/paper/2009/hash/5751ec3e9a4feab575962e78e006250d-Abstract.html>.
- Epoch AI. Data on AI models, 2026. URL <https://epoch.ai/data/ai-models>. Accessed September 15, 2026.
- Flax. NNX linear layers: Implementation and initialization, n.d. URL https://flax.readthedocs.io/en/latest/_modules/flax/nnx/nn/linear.html. Accessed September 15, 2026.
- K. He, X. Zhang, S. Ren, and J. 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. doi: 10.1109/ICCV.2015.123. URL <https://arxiv.org/abs/1502.01852>.
- J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models, 2020. URL <https://arxiv.org/abs/2001.08361>.
- Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer, 1998. URL <https://bottou.org/papers/lecun-98x>.
- J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018. URL <https://arxiv.org/abs/1711.00165>.
- R. M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer-Verlag, New York, 1996. URL <https://glizen.com/radfordneal/bnn.book.html>.
- PyTorch. Linear layer documentation, n.d. URL <https://docs.pytorch.org/docs/stable/generated/torch.nn.Linear.html>. Accessed September 15, 2026.
- G. Yang, E. J. Hu, I. Babuschkin, S. Sidor, X. Liu, D. Farhi, N. Ryder, J. Pachocki, W. Chen, and J. Gao. Tensor programs V: Tuning large neural networks via zero-shot hyperparameter transfer, 2022. URL <https://arxiv.org/abs/2203.03466>.