Double Descent Demystified: Identifying, Interpreting & Ablating the Sources of a Deep Learning Puzzle

Rylan Schaeffer[1], Mikail Khona[2], Zachary Robertson[1], Akhilan Boopathy[3], Kateryna Pistunova[4], Jason W. Rocks[5], Ila Rani Fiete[6], and Oluwasanmi Koyejo[1]

[1]Computer Science, Stanford University
[2]Physics, Massachusetts Institute of Technology
[3]EECS, Massachusetts Institute of Technology
[4]Physics, Stanford University
[5]Physics, Boston University
[6]Brain & Cognitive Sciences, Massachusetts Institute of Technology

March 2023

Abstract

Double descent is a surprising phenomenon in machine learning, in which as the number of model parameters grows relative to the number of data, test error drops as models grow ever larger into the highly overparameterized (data undersampled) regime. This drop in test error flies against classical learning theory on overfitting and has arguably underpinned the success of large models in machine learning. This non-monotonic behavior of test loss depends on the number of data, the dimensionality of the data and the number of model parameters. Here, we briefly describe double descent, then provide an explanation of why double descent occurs in an informal and approachable manner, requiring only familiarity with linear algebra and introductory probability. We provide visual intuition using polynomial regression, then mathematically analyze double descent with ordinary linear regression and identify three interpretable factors that, when simultaneously all present, together create double descent. We demonstrate that double descent occurs on real data when using ordinary linear regression, then demonstrate that double descent does not occur when any of the three factors are ablated. We use this understanding to shed light on recent observations in nonlinear models concerning superposition and double descent. Code is publicly available.

1 What is double descent?

Double descent is a phenomenon in machine learning describing the key observation that many classes of models can, under relatively broad conditions, exhibit seemingly perplexing changes in test loss as a function of three parameters: the number of data, the dimensionality of the data and the number of parameters in the model. For instance, as the number of model parameters increases, the test loss can fall, then rise, then fall again (Fig. 1). What causes such behavior?

Double descent has a rich history, both empirically and analytically; for a non-exhaustive list, see [13, 2, 19, 11, 15, 2, 1, 10, 6]. The term "double descent" was coined by [3], and some of our favorite papers on the topic include [16, 17, 18]. One important note is that not every dataset and model pair exhibits *two* descents; under different settings, there can be one, three or more descents [12, 5].

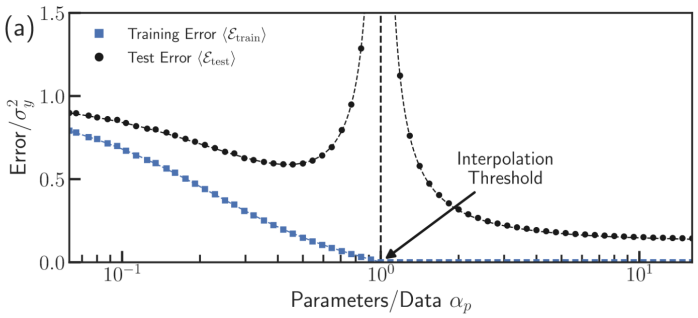

Figure 1: Double descent: test error falls, rises, then falls as a ratio of parameters to data. Fig. 3A from [18].

Our goal in this tutorial is to explain why double descent occurs in an approachable manner, without resorting to advanced tools oftentimes employed to analyze double descent such as random matrix theory or statistical physics. To accomplish this, we provide (1) visual intuition via polynomial regression, (2) mathematical analysis using ordinary linear regression, (3) empirical evidence from ordinary linear regression on real (tabular) data and (4) novel insights for nonlinear neural networks. To the best of our knowledge, we are the first to take this approach. Although we focus on regression tasks, our insights hold more generally.

2 Notation and Terminology

Consider a supervised dataset of N training data for regression:

$$\mathcal{D} \overset{\text{def}}{=} \Big\{ (\vec{x}_n, y_n) \Big\}_{n=1}^{N}$$

with covariates $\vec{x}_n \in \mathbb{R}^D$ and targets $y_n \in \mathbb{R}$. We'll sometimes use matrix-vector notation to refer to our training data, treating the features $\vec{x}_n$ as row vectors:

$$X \overset{\text{def}}{=} \begin{bmatrix} -\vec{x}_1 - \\ \vdots \\ -\vec{x}_N - \end{bmatrix} \in \mathbb{R}^{N \times D} \qquad Y \overset{\text{def}}{=} \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^{N \times 1}$$

In general, our goal is to use our training dataset $\mathcal{D}$ find a function $f : \mathcal{X} \to \mathcal{Y}$ that makes:

$$f(x) \approx y$$

In the setting of ordinary linear regression, we assume that f is a linear function i.e. $f(\vec{x}) = \vec{x} \cdot \vec{\beta}$, meaning our goal is to find (estimate) linear parameters $\hat{\vec{\beta}} \in \mathbb{R}^D$ that make:

$$\vec{x} \cdot \vec{\beta} \approx y$$

Of course, our real goal is to hopefully find a function that generalizes well to new data. As a matter of terminology, there are typically three key parameters:

1. The number of model parameters P
2. The number of training data N

3. The dimensionality of the data D

We say that a model is *overparameterized* (a.k.a. underconstrained) if $N < P$ and *underparameterized* (a.k.a. overconstrained) if $N > P$. The *interpolation threshold* refers to where $N = P$, because when $P \geq N$, the model can perfectly interpolate the training points.

3 Visual Intuition from Polynomial Regression

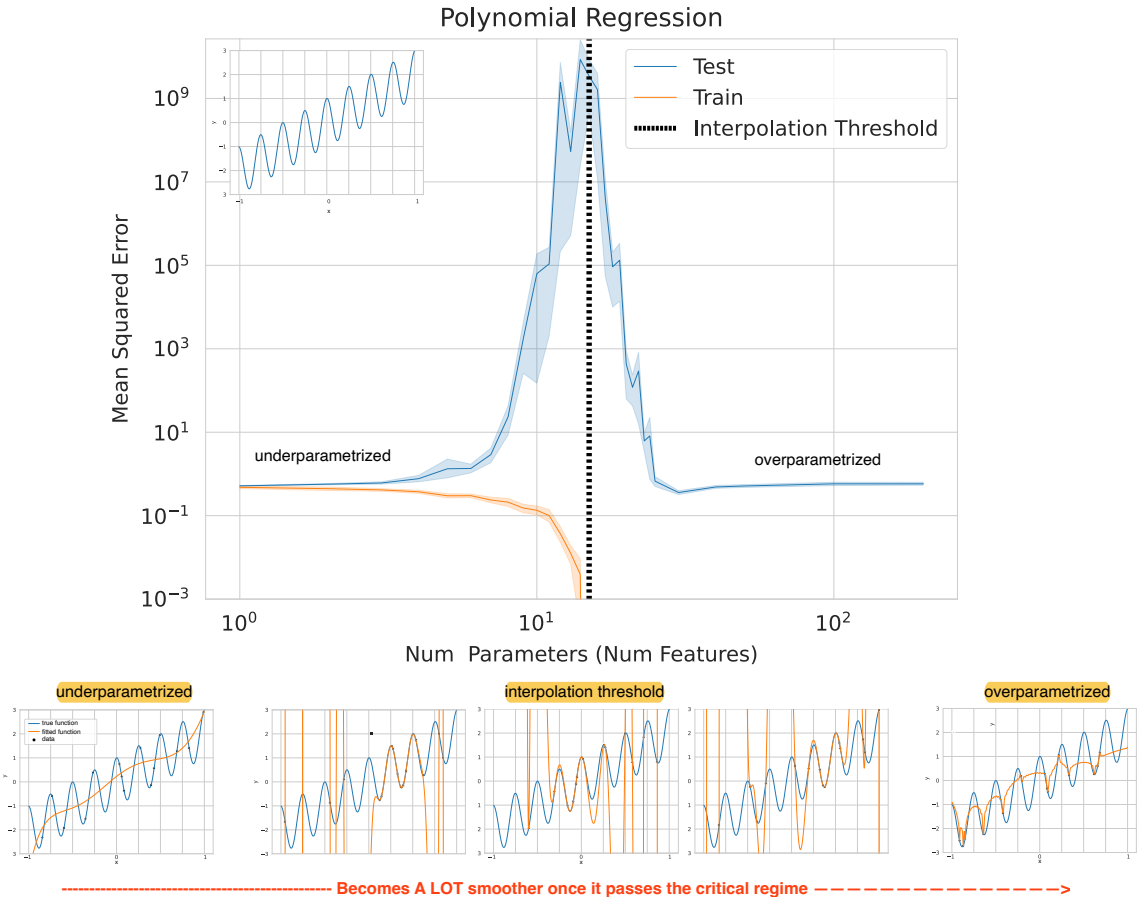

Figure 2: **Intuition for double descent from polynomial regression.** Top: Polynomial regression displays double descent. Bottom: When *underparameterized*, the model is unable to capture finer-grained features in the training data, meaning bias is large but variance is small. As the interpolation threshold is approached, the training data can be fit exactly, meaning bias is small; however, the particular realization of the training data significantly affects the learnt function, meaning variance is large. When *overparameterized*, the model can exactly fit the training data, meaning bias is again small, but the model is also regularized towards a small-norm solution, making variance small.

To offer visual intuition for the cause of double descent, we turn to polynomial regression. Concretely, suppose we wish to predict $y \in \mathbb{R}$ from $x \in [-1, 1]$, where the true (unknown) relationship is:

$$y(x) = 2x + \cos(25x)$$

In polynomial regression, we take the approach of mapping each datum x to a P-dimensional space

(corresponding to the P parameters) by using the following "feature" map $\vec{\phi}_P : \mathbb{R}^1 \to \mathbb{R}^P$:

$$\vec{\phi}_P : x \to \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_P(x) \end{bmatrix} \in \mathbb{R}^P$$

where ϕ_i denotes some polynomial[1]. We'll then fit a linear model using parameters $\vec{\beta}_P \in \mathbb{R}^P$:

$$y \approx \vec{\phi}_P(x) \cdot \vec{\beta}_P$$

We show the results of sweeping the number of parameters P (= the number of polynomials = the number of features) from 1 to 200 (Fig. 2). When $P = 1$, we can fit a line to our data, which is insufficiently expressive to capture the true relationship between x and y; thus, the model has bias, but low variance. As the number of parameters P increases towards the number of training data N, the model can more accurately fit the training data but at the cost of inducing "wiggles" that depend on the particular realization of the training data and that incur a high test mean squared error; the bias decreases but the variance diverges. As the number of parameters P increases beyond the number of training data N, the regression remains sufficiently expressive to exactly fit the training data, meaning it has low bias, but the model fitting process prefers solutions with smaller norm that "wiggle" less, meaning it has low variance as well. Next, we mathematically analyze under what conditions this double-descent phenomenon occurs.

4 Mathematical Intuition from Ordinary Linear Regression

To offer an intuitive yet quantitative understanding of double descent, we turn to ordinary linear regression. Recall that in linear regression, the number of fit parameters P must equal the dimension D of the covariates; consequently, rather than thinking about changing the number of parameters P, we'll instead think about changing the number of data N. *Because double descent is fundamentally about the ratio of number of parameters P to number of data N*, varying the number of data is as valid an approach as varying the number of parameters is. To understand where and why double descent occurs in linear regression, we'll study how linear regression behaves in the two parameterization regimes.

If the regression is *underparameterized*, we estimate the linear relationship between the covariates $\vec{x}_n$ and the target y_n by solving the classical least-squares minimization problem:

$$\hat{\vec{\beta}}_{under} \overset{\text{def}}{=} \arg\min_{\vec{\beta}} \frac{1}{N} \sum_n ||\vec{x}_n \cdot \vec{\beta} - y_n||_2^2 = \arg\min_{\vec{\beta}} ||X\vec{\beta} - Y||_F^2$$

The solution to this underparameterized optimization problem is the well-known ordinary least squares estimator that uses the second moment matrix $X^T X$:

$$\hat{\vec{\beta}}_{under} = (X^T X)^{-1} X^T Y$$

is this the pseudoinverse?

If the model is *overparameterized*, the above optimization problem is ill-posed since there are infinitely many solutions; this is because we have fewer constraints than parameters. Consequently, we need to choose a different (constrained) optimization problem:

$$\hat{\vec{\beta}}_{over} \overset{\text{def}}{=} \arg\min_{\vec{\beta}} ||\vec{\beta}||_2^2 \quad \text{s.t.} \quad \forall n \in \{1, ..., N\} \quad \vec{x}_n \cdot \vec{\beta} = y_n$$

[1] In our simulations, we choose ϕ_i to be the i-th Legendre polynomial

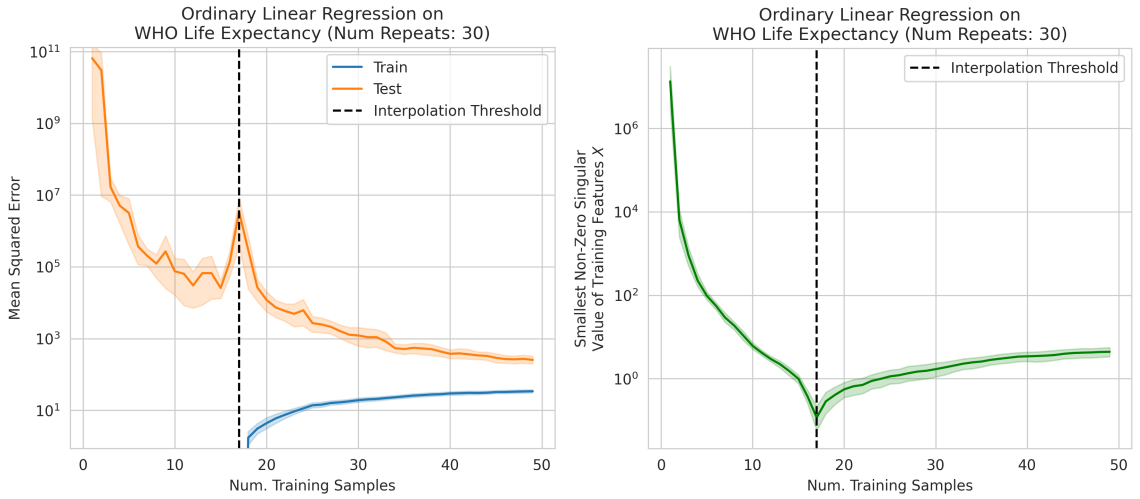

Figure 3: Left: Double descent occurs in ordinary linear regression on the World Health Organization's Life Expectancy dataset. Right: Double descent occurs when three quantities simultaneously grow extreme. One of the three is small-but-nonzero singular values in the training features X. The smallest non-zero singular value of X (probabilistically) obtains its lowest value when the number of parameters P equals the number of data N (the *interpolation threshold*), approaching from either the *overparameterized regime* ($P > N$; left) or the *underparameterized regime* ($P < N$; right).

The singular values are the absolute values of the eigenvalues of a normal matrix

This constrained optimization problem asks for the smallest parameters $\vec{\beta}$ that guarantee $\vec{x}_n \cdot \vec{\beta} = y_n$ for all training data. One reason why we choose this optimization problem is that it is the optimization problem that gradient descent implicitly minimizes (App. A). The solution to this optimization problem is less well-known and instead uses the so-called Gram matrix $XX^T \in \mathbb{R}^{N \times N}$:

$$\hat{\vec{\beta}}_{over} = X^T(XX^T)^{-1}Y$$

One way to see why the Gram matrix appears is via constrained optimization. Define the Lagrangian with Lagrange multipliers $\vec{\lambda} \in \mathbb{R}^N$:

$$\mathcal{L}(\vec{\beta}, \vec{\lambda}) \stackrel{\text{def}}{=} ||\vec{\beta}||_2^2 + \vec{\lambda}^T(Y - X\vec{\beta})$$

Differentiating with respect to both the parameters and the Lagrange multipliers yields:

$$\begin{aligned}
\nabla_{\vec{\beta}} \mathcal{L}(\vec{\beta}, \vec{\lambda}) = \vec{0} = 2\hat{\vec{\beta}} - X^T\vec{\lambda} &\Rightarrow \hat{\vec{\beta}}_{over} = \frac{1}{2}X^T\vec{\lambda} \\
\nabla_{\vec{\lambda}} \mathcal{L}(\beta, \lambda) = \vec{0} = Y - X\hat{\vec{\beta}}_{over} &\Rightarrow Y = \frac{1}{2}XX^T\vec{\lambda} \\
&\Rightarrow \vec{\lambda} = 2(XX^T)^{-1}Y \\
&\Rightarrow \hat{\vec{\beta}}_{over} = X^T(XX^T)^{-1}Y
\end{aligned}$$

Here, we are able to invert the Gram matrix because it is full rank in the overparametrized regime. After fitting its parameters, the model will make the following predictions for given test point $\vec{x}_{test}$:

$$\begin{aligned}
\hat{y}_{test,under} &= \vec{x}_{test} \cdot \hat{\vec{\beta}}_{under} = \vec{x}_{test} \cdot (X^TX)^{-1}X^TY \\
\hat{y}_{test,over} &= \vec{x}_{test} \cdot \hat{\vec{\beta}}_{over} = \vec{x}_{test} \cdot X^T(XX^T)^{-1}Y
\end{aligned}$$

Hidden in the above equations is an interaction between three quantities that can, when all grow extreme, create double descent. To reveal the three quantities, we'll rewrite the regression targets by introducing a slightly more detailed notation. Unknown to us, there are some ideal linear parameters $\vec{\beta}^* \in \mathbb{R}^P = \mathbb{R}^D$ that truly minimize the test mean squared error. We can write any regression target as the inner product of the data $\vec{x}_n$ and the ideal parameters β^*, plus an additional error term e_n that is an "uncapturable" residual from the "perspective" of the model class:

$$y_n = \vec{x}_n \cdot \vec{\beta}^* + e_n$$

In matrix-vector form, we will equivalently write:

$$Y = X\vec{\beta}^* + E$$

with $E \in \mathbb{R}^{N \times 1}$. To be clear, we are *not* imposing assumptions on the model or data. Rather, we are introducing notation to express that there are (unknown) ideal linear parameters, and possibly residuals that even the ideal model might be unable to capture; these residuals could be random noise or could be fully deterministic patterns that this particular model class cannot capture. Using this new notation, we rewrite the model's predictions to show how the test datum's features $\vec{x}_{test}$, training data's features X and training data's regression targets Y interact. In the underparameterized regime:

$$\begin{aligned}
\hat{y}_{test,under} &= \vec{x}_{test} \cdot (X^T X)^{-1} X^T Y \\
&= \vec{x}_{test} \cdot (X^T X)^{-1} X^T (X\beta^* + E) \\
&= \vec{x}_{test} \cdot (X^T X)^{-1} X^T X \beta^* + \vec{x}_{test} \cdot (X^T X)^{-1} X^T E \\
&= \underbrace{\vec{x}_{test} \cdot \beta^*}_{\stackrel{\text{def}}{=} y^*_{test}} + \vec{x}_{test} \cdot (X^T X)^{-1} X^T E \\
\hat{y}_{test,under} - y^*_{test} &= \vec{x}_{test} \cdot (X^T X)^{-1} X^T E
\end{aligned}$$

This equation is important, but opaque. To extract the intuition, we will replace X with its singular value decomposition[2] $X = U\Sigma V^T$ to reveal how different quantities interact. Let $R \stackrel{\text{def}}{=} rank(X)$ and let $\sigma_1 > \sigma_2 > ... > \sigma_R > 0$ be X's (non-zero) singular values. Recalling $E \in \mathbb{R}^{N \times 1}$, we can decompose the (underparameterized) prediction error $\hat{y}_{test,under} - y^*_{test}$ along the orthogonal singular modes:

$$\hat{y}_{test,under} - y^*_{test} = \vec{x}_{test} \cdot V\Sigma^+ U^T E = \sum_{r=1}^{R} \frac{1}{\sigma_r} (\vec{x}_{test} \cdot \vec{v}_r)(\vec{u}_r \cdot E)$$

In the overparameterized regime, our calculations change slightly:

$$\begin{aligned}
\hat{y}_{test,over} &= \vec{x}_{test} \cdot X^T (XX^T)^{-1} Y \\
&= \vec{x}_{test} \cdot X^T (XX^T)^{-1} (X\beta^* + E) \\
&= \vec{x}_{test} \cdot X^T (XX^T)^{-1} X\beta^* + \vec{x}_{test} \cdot X^T (XX^T)^{-1} E \\
\hat{y}_{test,over} - \underbrace{\vec{x}_{test} \cdot \beta^*}_{\stackrel{\text{def}}{=} y^*_{test}} &= \vec{x}_{test} \cdot X^T (XX^T)^{-1} X\beta^* - \vec{x}_{test} \cdot I_D \beta^* + \vec{x}_{test} \cdot (X^T X)^{-1} X^T E \\
\hat{y}_{test,over} - y^*_{test} &= \vec{x}_{test} \cdot (X^T (XX^T)^{-1} X - I_D)\beta^* + \vec{x}_{test} \cdot (X^T X)^{-1} X^T E
\end{aligned}$$

[2]For those unfamiliar with the SVD, any real-valued X can be decomposed into the product of three matrices $X = U\Sigma V^T$ where U and V are both orthonormal matrices and Σ is diagonal; intuitively, any linear transformation can be viewed as composition of first a rotoreflection, then a scaling, then another rotoreflection. Because Σ is diagonal and because U, V are orthonormal matrices, we can equivalently write $X = U\Sigma V^T$ in vector-summation notation as a sum of rank-1 outer products $X = \sum_{r=1}^{rank(X)} \sigma_r u_r v_r^T$. Each term in the sum is referred to as a "singular mode", akin to eigenmodes.

If we again replace X with its SVD USV^T, we can again simplify $\vec{x}_{test} \cdot (X^T X)^{-1} X^T E$. This yields our final equations for the prediction errors.

$$\hat{y}_{test,over} - y^*_{test} = \vec{x}_{test} \cdot (X^T (XX^T)^{-1} X - I_D)\beta^* + \sum_{r=1}^{R} \frac{1}{\sigma_r} (\vec{x}_{test} \cdot \vec{v}_r)(\vec{u}_r \cdot E)$$

$$\hat{y}_{test,under} - y^*_{test} = \sum_{r=1}^{R} \frac{1}{\sigma_r} (\vec{x}_{test} \cdot \vec{v}_r)(\vec{u}_r \cdot E)$$

What is the discrepancy between the underparameterized prediction error and the overparameterized prediction error, and from where does the discrepancy originate? The overparameterized prediction error $\hat{y}_{test,over} - y^*_{test}$ has the extra term $\vec{x}_{test} \cdot (X^T (XX^T)^{-1} X - I_D)\beta^*$. To understand where this term originates, recall that our goal is to understand how fluctuations in the features $\vec{x}$ correlate with fluctuations in the targets y. In the overparameterized regime, there are more parameters than there are data. Consequently, for N data points in $D = P$ dimensions, the model can "see" fluctuations in at most N dimensions, but has no "visibility" into fluctuations in the remaining $P - N$ dimensions. This causes information about the optimal linear relationship $\vec{\beta}^*$ to be lost, which in turn increases the overparameterized prediction error $\hat{y}_{test,over} - y^*_{test}$. Statisticians call this term $\vec{x}_{test} \cdot (X^T (XX^T)^{-1} X - I_D)\beta^*$ the "bias". The other term (the "variance") is what causes double descent:

$$\sum_{r=1}^{R} \frac{1}{\sigma_r} (\vec{x}_{test} \cdot \vec{v}_r)(\vec{u}_r \cdot E) \tag{1}$$

Eqn. 1 is critical. It reveals that our test prediction error (and thus, our test squared error!) will depend on an interaction between 3 quantities:

1. How much the *training features* X vary in each direction; more formally, the inverse (non-zero) singular values of the *training features* X: $\frac{1}{\sigma_r}$

   Training features are the underlying structure of the training distribution

   The inverse of the singular values of the training features means that that directions of variance in the data dimension that are the LEAST variant, have the largest imact. This makes sense as the smallest parts of the dim variance are the hardest to capture as they decrease the loss the least, and there's very little informatin gain/reward from modeling these dimensions

2. How much, and in which directions, the *test features* $\vec{x}_{test}$ vary relative to the *training features* X; more formally: how $\vec{x}_{test}$ projects onto X's right singular vectors V: $\vec{x}_{test} \cdot \vec{v}_r$

   How the axis of the training data map onto the eigenvectors of V

   Relationship between the training and test data structure. (or how similar they are)

3. How well the *best possible model* can correlate the variance in the *training features* X with the *training regression targets* Y; more formally: how the residuals E of the best possible model in the model class (i.e. insurmountable "errors" from the "perspective" of the model class) project onto X's left singular vectors U: $\vec{u}_r \cdot E$

Here, we use the terminology "vary" and "variance", suggesting a connection to the statistical notion of variance. There is indeed one[3], although we slightly abuse terminology! These three quantities,

[3] If we had centered the training features X to form $\bar{X}$, with corresponding SVD $\bar{X} = \bar{U}\bar{\Sigma}\bar{V}^T$, then $\bar{X}^T\bar{X}$ would be the empirical covariance matrix, and its eigendecomposition would be:

$$\bar{X}^T\bar{X} = \bar{V}\bar{\Sigma}^2\bar{V}^T$$

Each right singular vector of $\bar{X}$ would be an orthogonal axis of variation, with the variance along each direction given by the squared singular values of $\bar{X}$. However, because we don't center the features, $X^T X$ is the empirical second moment matrix, not the empirical covariance matrix. Thus, when we refer to "vary" and "variance", we are slightly abusing terminology, but the intuition - how much the features are wiggling, and whether the wiggling is correlated with the regression targets - is the right way to understand the concepts. We could center our data features to refer to the variance, but we felt doing so might be pedagogically confusing since centering is not related to double descent.

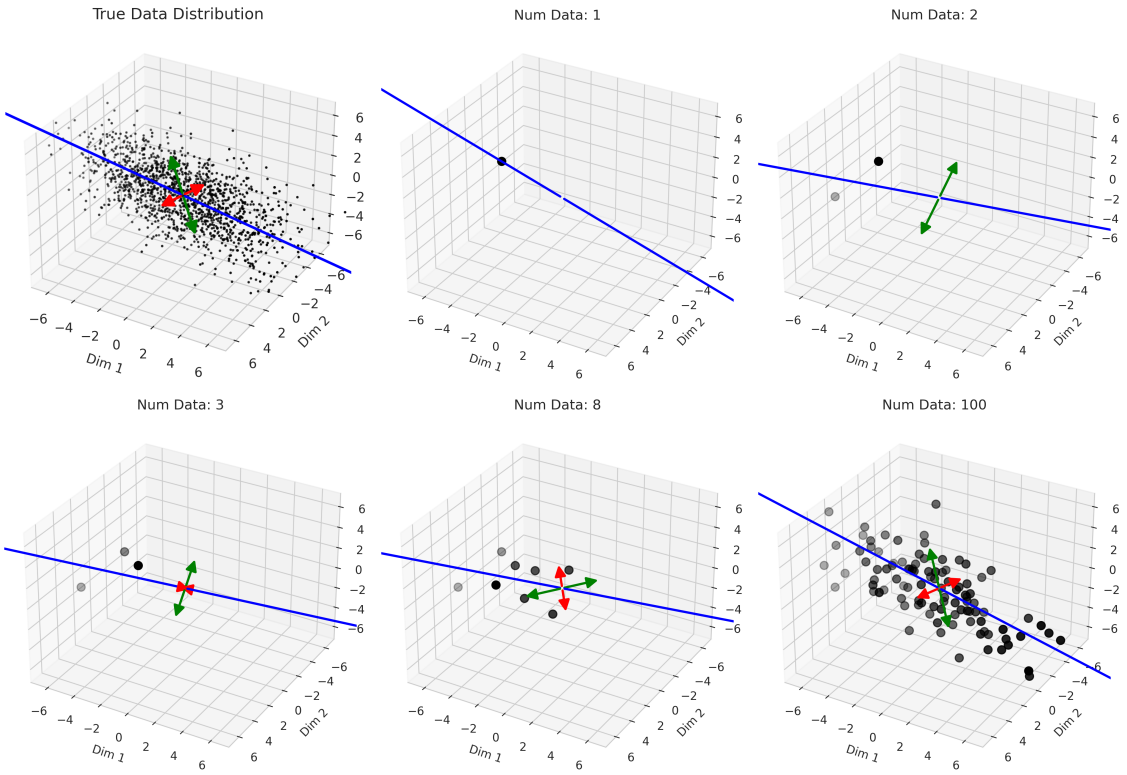

Figure 4: **Geometric intuition for why the smallest non-zero singular value reaches its lowest value when approaching the interpolation threshold.** As one nears the interpolation threshold (here, $D = P = N = 3$), N training data are unlikely to vary substantially in P orthogonal directions, meaning at least one orthogonal direction is likely to have small variance. If fewer data are observed, then the trailing directions have exactly zero variance, and if more data are observed, then the additional data reveal variance along these trailing directions.

multiplied, determine how much the r-th singular mode contributes to the prediction error.

Double descent occurs when the training features vary minorly in a singular direction but the test features vary largely

Double descent occurs when these three quantities grow extreme: (i) the *training features* contain small-but-nonzero variance in some singular direction(s), (ii) from the "perspective" of the model class, residual errors in the *training features and targets* have large projection along this singular mode, and (iii) the *test features* vary significantly along this singular mode. When (i) and (ii) co-occur, this means the model's parameters along this mode are likely incorrect. Then, when (iii) is added to the mix by a test datum $\vec{x}_{test}$ with a large projection along this mode, the model is forced to extrapolate significantly beyond what it saw in the training data, in a direction where the training data had an error-prone relationship between its training predictions and the training regression targets, using parameters that are likely wrong. As a consequence, the test squared error explodes!

Why does this explosion happen near the interpolation threshold? The answer is that the first factor becomes more likely to occur when approaching the interpolation threshold from either parameterization regime. The reason why the smallest non-zero singular value is (probabilistically) likely to reach its lowest value at the interpolation threshold is a probabilistic one, based on the Marchenko–Pastur distribution from random matrix theory. Because the Marchenko–Pastur distribution is rather technical, we instead focus on gaining intuition by thinking about how much variance we've seen along each orthogonal direction in the data feature space (Fig. 4).

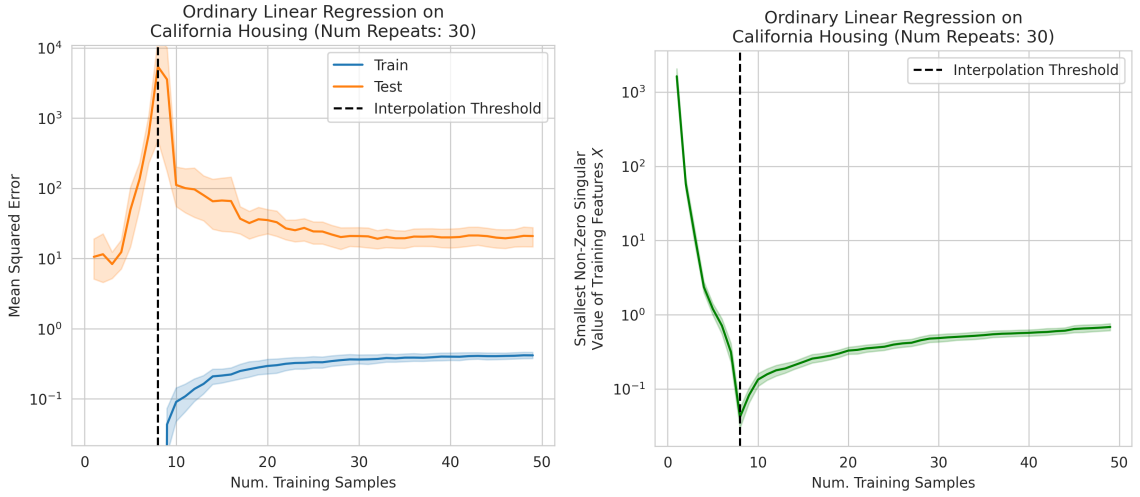

Figure 5: Double Descent in Ordinary Linear Regression on California Housing Dataset.

The smallest non-zero singular value reflects the least amount of variance in any direction captured by the data points (and thus, the least amount of information). This increase in the smallest singular value means that there is additional variance (information) we can obtain from more data samples

Singular values in SVD tell us how much variance each principal component (or direction) explains in the data. Larger singular values correspond to directions with more variance.

Good explanation of why double descent occurs.

Suppose we're given a single training datum $\vec{x}_1$. So long as this datum isn't exactly zero, that datum varies in a single direction, meaning we gain information about the data distribution's variance in that direction. Of course, the variance in all orthogonal directions is exactly 0, which the linear regression fit will ignore. Now, suppose we're given a second training datum $\vec{x}_2$. Again, so long as this datum isn't exactly zero, that datum varies, but now, some fraction of $\vec{x}_2$ might have a positive projection along $\vec{x}_1$; if this happens (and it likely will, since the two vectors are unlikely to be exactly orthogonal), the shared direction of the two vectors gives us *more* information about the variance in this shared direction, but gives us *less* information about the second orthogonal direction of variation. This means that the training data's smallest non-zero singular value after 2 samples is probabilistically smaller than after 1 sample. As we gain more training data, thereby approaching the interpolation threshold, the probability that each additional datum has large variance in a new direction orthogonal to all previously seen directions grows increasingly unlikely. At the interpolation threshold, where $N = P = D$, in order for the N-th datum to avoid adding a small-but-nonzero singular value to the training data, two properties must hold: (1) there must be one dimension that none of the preceding $N - 1$ training data varied in, and (2) the N-th datum needs to vary significantly in this single dimension. That's pretty unlikely! As we move beyond the interpolation threshold, the variance in each covariate dimension becomes increasingly clear, and the smallest non-zero singular values moves away from 0. This is displayed visually in Fig. 4.

5 Empirical Evidence on Real Data with Linear Regression

Does our claim that ordinary linear regression can exhibit double descent hold empirically? We show that it indeed does, using three datasets: WHO Life Expectancy, California Housing, Diabetes. These three datasets were randomly selected on the basis of being easily accessible, e.g., through sklearn [14]. All display a sharp spike in test mean squared error at the interpolation threshold (Left panels of Figs. 3, 5, 6). We additionally substantiate our previous claim that the smallest non-zero singular value of the training features X obtains its lowest value as one nears the interpolation threshold (Right panels of Figs. 3, 5, 6). Code is publicly available.

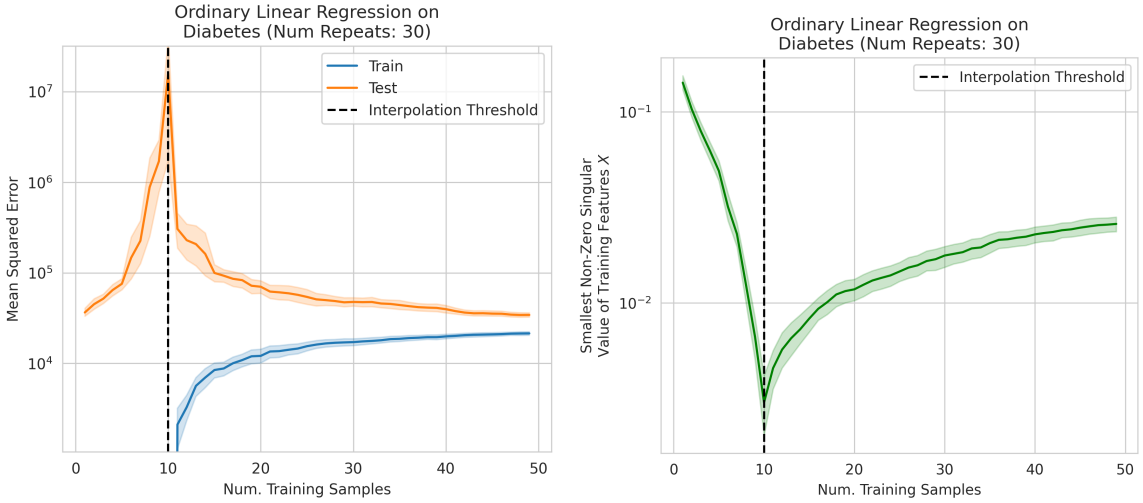

Figure 6: Double Descent in Ordinary Linear Regression on Diabetes Dataset.

6 When Does Double Descent Not Occur?

Double descent will not occur if any of the three factors are absent. What could cause that?

- *Small-but-nonzero singular values do not appear in the training data features.* One way to accomplish this is by switching from ordinary linear regression to ridge regression, which effectively adds a gap separating the smallest non-zero singular value from 0.
- *The test datum does not vary in different directions than the training features.* If the test datum lies entirely in the subspace of just a few of the leading singular directions, then double descent is unlikely to occur.
- *The best possible model in the model class makes no errors on the training data.* For instance, suppose we use a linear model class on data where the true relationship is a noiseless linear one. Then, at the interpolation threshold, we will have $D = P$ data, $P = D$ parameters, our line of best fit will exactly match the true relationship, and no double descent will occur.

To confirm our understanding, we causally test the predictions of when double descent will not occur by ablating each of the three factors individually. Specifically, we do the following:

1. No Small Singular Values in Training Features: As we run the ordinary linear regression fitting process, as we sweep the number of training data, we also sweep different singular value cutoffs and remove all singular values of the training features X below the cutoff.
2. Test Features Lie in the Training Features Subspace: As we run the ordinary linear regression fitting process, as we sweep the number of training data, we project the test features $\vec{x}_{test}$ onto the subspace spanned by the training features X singular modes.
3. No Residual Errors in the Optimal Model: We first use the entire dataset to fit a linear model $\vec{\beta}^*$, then replace Y with $X\vec{\beta}^*$ and y^*_{test} with $\vec{x}_{test} \cdot \vec{\beta}^*$ to ensure the true relationship is linear. We then rerun our typical fitting process, sweeping the number of training data.

We first conduct experiments on a synthetic dataset in a student-teacher setup, and find that causally ablating each of the three factors prevents double descent from occurring (Fig. 7, top row). Next, we apply the same ablations to real world datasets (California Housing, Diabetes, WHO Life Expectancy) and find in all three that removing any of the three factors prevents double descent (Fig. 7, rows 2-5).

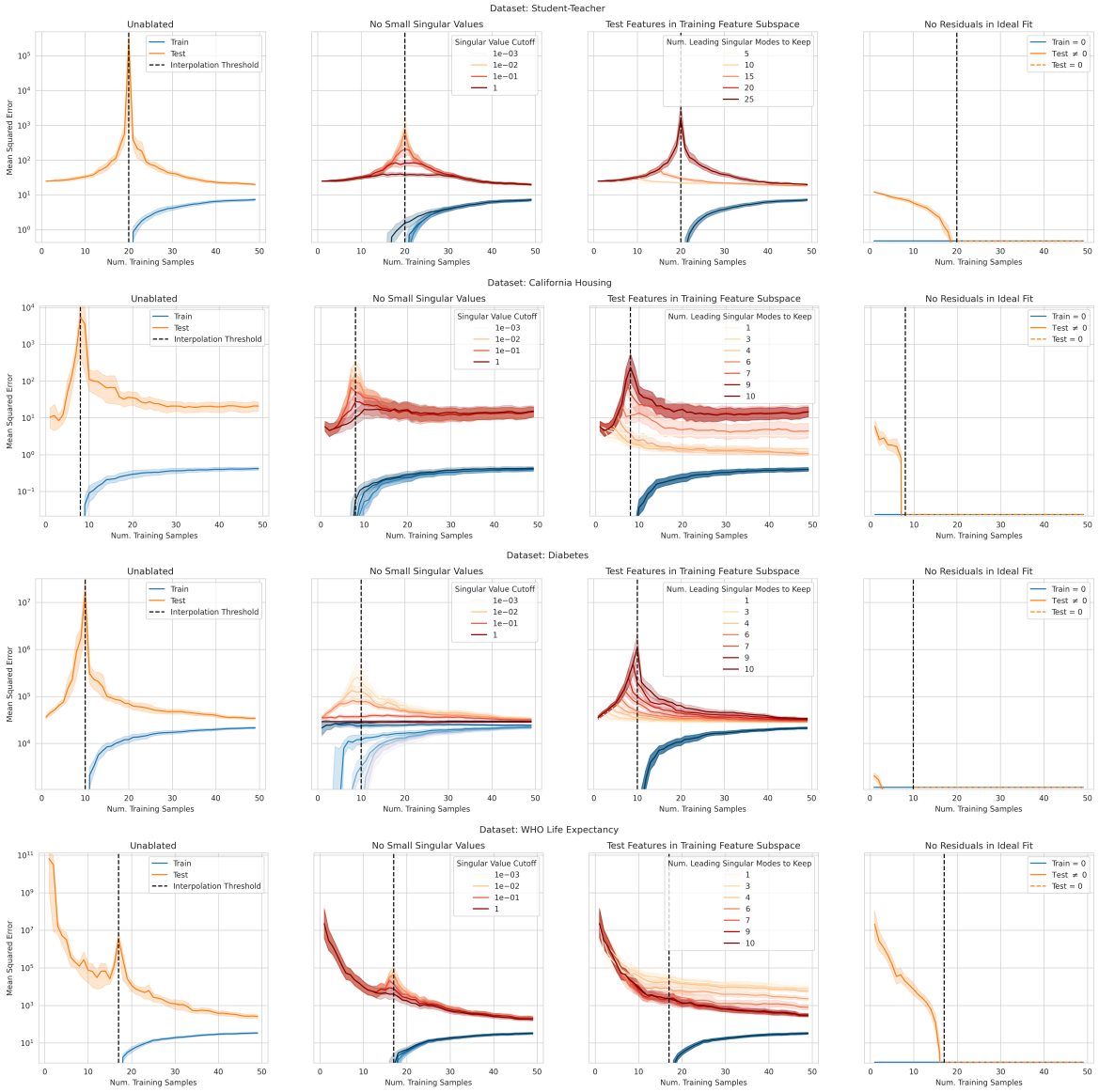

Figure 7: **Double descent will not occur if any of the three critical quantities are absent.** We demonstrate this via ablations (Sec. 6). Left to Right: Double descent appears in ordinary linear regression. Removing small singular values in the training features X prevents double descent. Preventing the test features $\vec{x}_{test}$ from varying in the trailing singular modes of the training features X prevents double descent. Ensuring that the optimal model in the model class has zero residual prediction errors E prevents double descent. Top to Bottom: Synthetic data generated in a Student-Teacher framework, California Housing dataset, Diabetes dataset, WHO Life Expectancy dataset

7 Intuition Extends to Nonlinear Models

Although we mathematically studied ordinary linear regression, the intuition for why double descent occurs extends to nonlinear models, including deep neural networks. For instance, we can conduct the same analysis we did for linear regression but instead for polynomial regression by considering the training covariates X projected into a (typically much higher dimensional) polynomial feature space:

$$\Phi_P(X) \stackrel{\text{def}}{=} \begin{bmatrix} -\vec{\phi}_P(\vec{x}_1)- \\ -\vec{\phi}_P(\vec{x}_2)- \\ \vdots \\ -\vec{\phi}_P(\vec{x}_N)- \end{bmatrix}$$

For certain classes of infinitely wide deep neural networks (e.g., as studied [8, 9, 4]), the output predictions of the network are equivalent to a linear regression performed on a particular set of features. Different infinite width limits correspond to different sets of features; for instance, one such limit (known as the Neural Network Gaussian Process limit [9]), studies the training covariates X represented in the penultimate layer of the deep neural network $\vec{f}_\theta(x)$:

$$F_\theta(X) \stackrel{\text{def}}{=} \begin{bmatrix} -\vec{f}_\theta(\vec{x}_1)- \\ -\vec{f}_\theta(\vec{x}_2)- \\ \vdots \\ -\vec{f}_\theta(\vec{x}_N)- \end{bmatrix}$$

For a concrete example about how our intuition can shed light on the behavior of nonlinear models, [7] recently discovered interesting properties of shallow nonlinear autoencoders: depending on the number of training data, (1) autoencoders either store data points or features, and (2) double descent occurs between these two regimes. Our tutorial helps explains the results, and also sheds light on two comments the authors make:

1. [7] write, "[Our work] suggests a naive mechanistic theory of overfitting and memorization: memorization and overfitting occur when models operate on "data point features" instead of "generalizing features". We expect this naive theory to be overly simplistic, but it seems possible that it's gesturing at useful principles!" Our tutorial hopefully clarifies that this choice of terminology ("data point features" vs. "generalizing features") can be made more precise. When overparameterized, the "data point features" are akin to the data-by-data Gram matrix $XX^T \in \mathbb{R}^{N\times N}$ and when underparameterized, the "generalizing features" are akin to the feature-by-feature second moment matrix $X^TX \in \mathbb{R}^{D\times D}$. Our tutorial hopefully shows that "data point features" can (and very often do) generalize, and that there is a deep connection between the two, e.g., their shared spectra.

2. [7] write, "It's interesting to note that we're observing double-descent in the absence of label noise. That is to say: the inputs and targets are exactly the same. Here, the "noise" arises from the lossy compression happening in the down projection." Our tutorial clarifies that noise in the sense of a random unpredictable quantity is *not* necessary to produce double descent. Rather, what is necessary is *residual errors from the perspective of the model class.* Those residual errors could be entirely deterministic, such as a nonlinear model attempting to fit a noiseless linear relationship.

8 Acknowledgements

We thank David K. Zhang, Matthew Sotoudeh, Victor Lecomte, Krista Opsahl-Ong, Aaron Scher, Max Lamparth, Zane Durante, Gabriel Mukobi, Brando Miranda and Laureline Logiaco for feedback.

References

[1] Ben Adlam and Jeffrey Pennington. Understanding double descent requires a fine-grained bias-variance decomposition. *Advances in neural information processing systems*, 33:11022–11032, 2020.

[2] Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020.

[3] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[4] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. In *International Conference on Machine Learning*, pages 1024–1034. PMLR, 2020.

[5] Lin Chen, Yifei Min, Mikhail Belkin, and Amin Karbasi. Multiple descent: Design your own generalization curve. *Advances in Neural Information Processing Systems*, 34:8898–8912, 2021.

[6] Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *The Annals of Statistics*, 50(2):949–986, 2022.

[7] Tom Henighan, Shan Carter, Tristan Hume, Nelson Elhage, Robert Lasenby, Stanislav Fort, Nicholas Schiefer, and Christopher Olah. Double descent in the condition number. *Transformer Circuits Thread*, 2023.

[8] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[9] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[10] Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and the double descent curve. *Communications on Pure and Applied Mathematics*, 75(4):667–766, 2022.

[11] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[12] Preetum Nakkiran, Prayaag Venkat, Sham Kakade, and Tengyu Ma. Optimal regularization can mitigate double descent. *arXiv preprint arXiv:2003.01897*, 2020.

[13] Manfred Opper. Statistical mechanics of learning: Generalization. *The handbook of brain theory and neural networks*, pages 922–925, 1995.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Tomaso Poggio, Gil Kur, and Andrzej Banburski. Double descent in the condition number. *arXiv preprint arXiv:1912.06190*, 2019.

[16] Jason W Rocks and Pankaj Mehta. The geometry of over-parameterized regression and adversarial perturbations. *arXiv preprint arXiv:2103.14108*, 2021.

[17] Jason W Rocks and Pankaj Mehta. Bias-variance decomposition of overparameterized regression with random linear features. *Physical Review E*, 106(2):025304, 2022.

[18] Jason W Rocks and Pankaj Mehta. Memorizing without overfitting: Bias, variance, and interpolation in overparameterized models. *Physical Review Research*, 4(1):013201, 2022.

[19] Stefano Spigler, Mario Geiger, Stéphane d'Ascoli, Levent Sagun, Giulio Biroli, and Matthieu Wyart. A jamming transition from under-to over-parametrization affects loss landscape and generalization. *arXiv preprint arXiv:1810.09665*, 2018.

A Why Gradient Descent Implicitly Regularizes

This is a sketch of why gradient descent implicitly regularizes. Suppose we have a model Xw for a vector of data $y \in \mathbb{R}^n$ and want to minimize the norm of the error,

$$L(w) = \|Xw - y\|_2^2 = \|e\|_2^2$$

where we introduce some short-hand notation. We use the gradient learning rule,

$$w(t+1) = w(t) - \eta X^T e(t)$$

$$\Rightarrow e(t+1) = e(t) - \eta X X^T e(t)$$

$$\Rightarrow e(t+1) = (I - \eta X X^T) e(t)$$

Each matrix satisfies $X \in \mathbb{R}^{n \times d_1}$ where n is the number of samples and d_1 is the dimension of each sample. In the overparameterized setting we have $d_1 > n$ and so XX^T will generically have full-rank and the error will go to zero.

^ n x n matrix

This lies in the difference between XX^T which appears here in the error analysis and $X^T X$ which appears in the solution. So we can have $XX^T \in \mathbb{R}^{n \times n}$ generically full-rank only if we have more parameters than there is data. On the other hand, we only have $X^T X$ full-rank if also it's satisfied that there is more data than parameters. This is important because in this case we can compute the pseudo-inverse easily. Generically, we can show that if we use gradient descent we have something like the following,

$$\underbrace{(X^T X)^{-1} X}_{\text{left inverse}} \quad \underbrace{X^{-1}}_{\text{inverse}} \quad \underbrace{X^T (X X^T)^{-1}}_{\text{right inverse}}$$

for the cases where we are under-parameterized, minimally parameterized, or over-parameterized to model the data.

So under gradient flow we'll suppose the parameters update according to,

$$\dot{w} = -\eta X^T e$$

$$w(0) = 0$$

Observe that the gradient $\dot{w}$ is invariantly in the span of X^T so we may conclude that $w(t)$ is always in the span of X^T. Generically, any solution in the over-parameterized setting is a global optimizer such that $Xw = y$. This means that the limit of the flow can be written as $w^* = X^T \alpha$ for some coefficient vector with the constraint that $Xw^* = y$. After some manipulations we find that,

$$y = Xw^* = XX^T \alpha$$

$$\Rightarrow \alpha = (XX^T)^{-1} y$$

$$\Rightarrow w^* = X^T (XX^T)^{-1} y = X^+ y$$

This means that the solution X^+ picked from gradient flow is the Moore-Penrose psuedoinverse. This can be defined as the matrix,

$$X^+ = \lim_{\lambda \to 0^+} X^T (XX^T + \lambda I)^{-1}$$

Also observe that there is a unique minimizer for the regularized problem,

$$\min_w L(w) + \lambda \|w\|_2^2$$

with value $w_\lambda = X^T (XX^T + \lambda I)^{-1} y$. Perhaps, $Xw = y$ has a set of solutions, but it is clear this set is convex so there is a unique minimum norm solution. On the other hand, each w_λ corresponds to a best solution with norm below v_λ, which is less than the minimum. However, we have $w^* = \lim_{\lambda \to 0^+} w_\lambda$ from continuity. Since w^* is an exact solution it can't have less than the minimum-norm and it is clear w^* can't have above the minimum-norm either since this is not the case for any of the w_λ. We conclude that gradient descent does indeed find the minimum norm solution.