Chapter 1
A Toy Model: Linear Regression

Roger Grosse

1 Some Motivating Phenomena

This is a class about understanding the dynamics of neural net training. We'll begin by analyzing some particular phenomena that people have been confused about in recent years. While the explanations are simpler than the ones we'll consider subsequently in the course, this lecture should give a sense for how we can understand neural net phenomena by analyzing simpler model systems. The phenomena we'll look at today are:

1. **The benefits of normalization/standardization.** It's beneficial (in fact, standard practice) to **normalize**, or **standardize**, the input dimensions to be zero mean and unit variance. This improves both the training speed and generalization.

2. **Double descent.** Introductory machine learning courses typically claim that as you increase the dimensionality of a model, the validation error first goes down (as the model fits the structure in the data) and then goes up (as it starts overfitting). But it's sometimes been observed that the validation error behaves *non-monotonically*, as shown in Figure 1. Why would increasing the size of the model *reduce* overfitting, and why doesn't this happen until the network gets very large? (This phenomenon was observed by Belkin et al. (2019).)

The first phenomenon is closely related to **batch normalization**, an operation that's a standard part of many deep learning architectures today. Batch norm has often been cited as the canonical example of deep learning "alchemy", whereby researchers stumbled upon a trick that happened to work really well, without any understanding of the underlying principles. But in fact, batch norm was carefully engineered based on principles that were well understood at the time. The main idea the batch norm inventors used to explain their method — but which was unnoticed by many of batch norm's critics — was exactly the analysis we'll give today of the benefits of input normalization. (We'll analyze batch norm itself in a later lecture.)

The first question to ask about any neural net phenomenon is: does it also happen for linear regression? Neural nets are complicated and nonlinear, and rarely admit a closed-form analysis. But for linear regression, many questions can be answered analytically. How fast do algorithms like gradient descent converge, and what do they converge to? How well does the solution generalize? Do things like this depend on seemingly insiginificant changes to the formulation of the problem? When we're talking about linear regression, questions like these can be reduced to linear algebra.

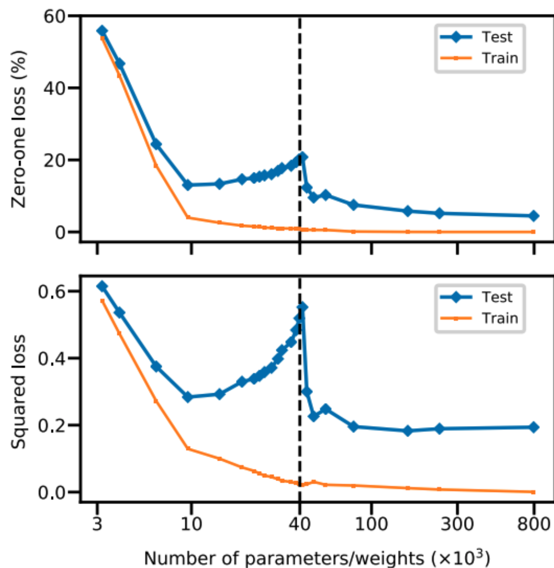

Figure 1: Figure from Belkin et al. (2019) illustrating the double descent phenomenon for neural nets. This shows the training and test error on (a subset of) MNIST for a fully connected network as the number of hidden units is varied (represented on the x-axis as the number of parameters). The dashed line represents the **interpolation threshold** where the number of parameters is just enough to memorize the training data.

2 Gradient Descent for Linear Regression

Recall that linear regression tries to model a scalar-valued target t as a function of an input vector $\mathbf{x}$, and the model is assumed to be linear in some feature representation $\boldsymbol{\phi}(\mathbf{x})$:

$$y = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + b. \tag{1}$$

It's sometimes notationally convenient to use a homogeneous coordinate representation, where a 1 is appended to the feature vector. I.e., we take $\breve{\boldsymbol{\phi}}(\mathbf{x}) = (\boldsymbol{\phi}(\mathbf{x})^\top \; 1)^\top$ and $\breve{\mathbf{w}} = (\mathbf{w}^\top \; b)^\top$, so that

$$y = \breve{\mathbf{w}}^\top \breve{\boldsymbol{\phi}}(\mathbf{x}). \tag{2}$$

We're given a finite training set $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$, and we fit the model by minimizing the mean squared error on the training set:

$$\begin{aligned}
\breve{\mathbf{w}}_\star &\in \arg\min_{\breve{\mathbf{w}}} \mathcal{J}(\breve{\mathbf{w}}) \\
&= \arg\min_{\breve{\mathbf{w}}} \frac{1}{2N} \sum_{i=1}^N (\breve{\mathbf{w}}^\top \breve{\boldsymbol{\phi}}(\mathbf{x}^{(i)}) - t^{(i)})^2 \\
&= \arg\min_{\breve{\mathbf{w}}} \frac{1}{2N} \|\breve{\boldsymbol{\Phi}} \breve{\mathbf{w}} - \mathbf{t}\|^2,
\end{aligned} \tag{3}$$

where $\breve{\boldsymbol{\Phi}}$ is a matrix whose ith row is the feature vector $\breve{\boldsymbol{\phi}}(\mathbf{x}^{(i)})$, and $\mathbf{t}$ is the vector of all the targets. Note that we use the notation $\in$ above, since

we don't know whether the arg min is unique (and in fact, it often won't be). If it's not unique, then which optimum we wind up in depends on the dynamics of training — one of the main motivations for studying the dynamics!

Observe that the cost function $\mathcal{J}(\breve{\mathbf{w}})$ is quadratic in $\breve{\mathbf{w}}$, and also that it's a convex quadratic, because it's a sum of squares and therefore nonnegative. In understanding the gradient descent dynamics, it's helpful to abstract away the regression problem and think about minimizing a convex quadratic objective:

$$\mathbf{w}_\star \in \arg\min_{\mathbf{w}} \tfrac{1}{2}\mathbf{w}^\top \mathbf{A}\mathbf{w} + \mathbf{b}^\top \mathbf{w} + c, \tag{4}$$

where $\mathbf{A}$ is a positive semidefinite matrix, $\mathbf{b}$ is a vector, and c is a scalar. You can check that the linear regression cost corresponds to $\mathbf{A} = \frac{1}{N}\breve{\mathbf{\Phi}}^\top \breve{\mathbf{\Phi}}$ and $\mathbf{b} = -\frac{1}{N}\breve{\mathbf{\Phi}}^\top \mathbf{t}$. The constant offset c doesn't affect the optimum or the optimization dynamics (because it doesn't affect the gradient), so we'll ignore it from here on.

The vector $\mathbf{b}$ has nothing to do with the bias parameter b.

2.1 Some Observations about Gradient Descent

Gradient descent is an iterative algorithm where each iteration updates the weights opposite the gradient direction:

$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}^{(k)}) \tag{5}$$

The optimization-minded reader might wonder why we're using a fixed α rather than a line search. A line search would indeed be better for minimizing a deterministic objective, but most neural net optimization is stochastic, so we don't have good line search methods. Hence, we use a fixed α.

The first question to ask about an update rule is: what are its **fixed points**, also called **stationary points**? I.e., for which values $\mathbf{w}^{(k)}$ does $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)}$? By setting the left-hand side to equal the right-hand side, we see that the fixed points for gradient descent are exactly the **critical points** of $\mathcal{J}$, i.e. those points where $\nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \mathbf{0}$. For general differentiable functions, critical points could include local maxima, local minima, or saddle points. However, if $\mathcal{J}$ is convex (as in linear regression), then any critical point corresponds to a global minimum of $\mathcal{J}$.

An important property of gradient descent is that it is **invariant to rigid transformations**. Rigid transformations include translations, rotations, and reflections, so we say that it is **translation invariant**, **rotation invariant**, etc. What we mean by this is that if you rotate the objective function and the initial weights, then the new gradient descent trajectory will be a rotation of the original trajectory. This should be intuitively obvious: you can determine the SGD updates for a 2D function by printing out the contour plot on a piece of paper and drawing arrows in the downhill direction. If you move the sheet of paper around on your desk, the direction of the arrow on the page doesn't change.

Mathematically, let $\mathbf{Q}$ be an orthogonal matrix and $\mathbf{t}$ be a vector. We define the transformation

$$\tilde{\mathbf{w}} = \mathcal{T}(\mathbf{w}) = \mathbf{Q}^\top(\mathbf{w}^{(0)} - \mathbf{t}), \tag{6}$$

Since $\mathbf{Q}$ is an arbitrary orthogonal matrix, we're showing invariance to both rotation and reflection. If we want only rotation, we can restrict it to have determinant 1.

which has the inverse transformation

$$\mathbf{w} = \mathcal{T}^{-1}(\tilde{\mathbf{w}}) = \mathbf{Q}\tilde{\mathbf{w}} + \mathbf{t}. \tag{7}$$

This can be seen as a **change-of-basis transformation**, or **reparameterization**: the transformed weights $\tilde{\mathbf{w}}$ simply re-express $\mathbf{w}$ in a coordinate system whose origin is $\mathbf{t}$ and whose (orthogonal) basis is given by the columns of $\mathbf{Q}$. The cost function can be re-expressed in the new coordinate system as follows:

$$\tilde{\mathcal{J}}(\tilde{\mathbf{w}}) = \mathcal{J}(\mathcal{T}^{-1}(\tilde{\mathbf{w}})) = \mathcal{J}(\mathbf{Q}\tilde{\mathbf{w}} + \mathbf{t}). \tag{8}$$

In order to show invariance, we need to show that the gradient descent trajectory in the transformed space is the transformation of the gradient descent trajectory in the original space. Mathematically, let $(\mathbf{w}^{(k)})_{k=0}^{\infty}$ denote the gradient descent trajectory in the original space starting from initialization $\mathbf{w}^{(0)}$, and $(\tilde{\mathbf{w}}^{(k)})_{k=0}^{\infty}$ denote the gradient descent trajectory in the transformed space starting from $\tilde{\mathbf{w}}^{(0)} = \mathcal{T}(\mathbf{w}^{(0)})$. We need to show that $\tilde{\mathbf{w}}^{(k)} = \mathcal{T}(\mathbf{w}^{(k)})$ for all k.

We can do this by induction. The base case is covered by our assumption that $\tilde{\mathbf{w}}^{(0)} = \mathcal{T}(\mathbf{w}^{(0)})$. For the inductive step, assume $\tilde{\mathbf{w}}^{(k)} = \mathcal{T}(\mathbf{w}^{(k)})$. By the Chain Rule for derivatives, we have:

$$\begin{aligned}\nabla \tilde{\mathcal{J}}(\tilde{\mathbf{w}}^{(k)}) &= \mathbf{Q}^\top \nabla \mathcal{J}(\mathbf{Q}\tilde{\mathbf{w}}^{(k)} + \mathbf{t}) \\ &= \mathbf{Q}^\top \nabla \mathcal{J}(\mathbf{w}^{(k)})\end{aligned}$$

Hence, the gradient descent update is:

$$\begin{aligned}\tilde{\mathbf{w}}^{(k+1)} &= \tilde{\mathbf{w}}^{(k)} - \alpha \nabla \tilde{\mathcal{J}}(\tilde{\mathbf{w}}^{(k)}) \\ &= \tilde{\mathbf{w}}^{(k)} - \alpha \mathbf{Q}^\top \nabla \mathcal{J}(\mathbf{w}^{(k)}) \\ &= \mathbf{Q}^\top (\mathbf{w}^{(k)} - \mathbf{t}) - \alpha \mathbf{Q}^\top \nabla \mathcal{J}(\mathbf{w}^{(k)}) \\ &= \mathcal{T}(\mathbf{w}^{(k+1)})\end{aligned}$$

We've just shown that gradient descent is invariant to rigid transformations.

Note: gradient descent is *not* invariant to arbitrary linear transformations, as we'll see later. Ask yourself where this proof breaks down if $\mathbf{Q}$ is replaced by an arbitrary invertible matrix $\mathbf{T}$. (Hint: the transformation has to be defined as $\mathcal{T}(\mathbf{w}) = \mathbf{T}^{-1}(\mathbf{w} - \mathbf{t})$, while the chain rule for derivatives still produces $\mathbf{T}^\top$.)

2.2 Closed Form Dynamics for Convex Quadratics

Now let's analyze gradient descent for the convex quadratic objective (Eqn. 4). Plugging this objective into Eqn. 5, we get the gradient descent update rule:

The analysis of convex quadratics is used in a large fraction of the course, not just as a tool for analyzing linear regression. So it is worth getting comfortable with this derivation.

$$\begin{aligned}\mathbf{w}^{(k+1)} &\leftarrow \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}^{(k)}) \\ &= \mathbf{w}^{(k)} - \alpha \left[\mathbf{A}\mathbf{w}^{(k)} + \mathbf{b}\right] \\ &= (\mathbf{I} - \alpha \mathbf{A})\mathbf{w}^{(k)} - \alpha \mathbf{b}\end{aligned} \tag{9}$$

Since the cost function is convex, the stationary points (if they exist) are global minima, and are given by:

$$\nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \mathbf{A}\mathbf{w} + \mathbf{b} = \mathbf{0}. \tag{10}$$

If the matrix $\mathbf{A}$ happens to be invertible, then this equation has a unique solution $\mathbf{w} = -\mathbf{A}^{-1}\mathbf{b}$, i.e. gradient descent has a unique fixed point corresponding to the unique global optimum. What happens if $\mathbf{A}$ isn't invertible? If $\mathbf{b}$ is not contained in the row space of $\mathbf{A}$, then Eqn. 10 has no solutions. If $\mathbf{b}$ is contained in the row space of $\mathbf{A}$ but $\mathbf{A}$ is non-invertible, then it has multiple solutions, i.e. the subspace given by $\{\mathbf{w}_0 + \Delta\mathbf{w} : \Delta\mathbf{w} \in \text{null space of } \mathbf{A}\}$, where $\mathbf{w}_0$ is any particular solution.

In linear algebra terminology, this is really an affine space. But in ML, we use *subspace* to mean *affine space* and *linear* to mean *affine.* E.g., it's *linear regression*, not *affine regression.* Hopefully it will all be clear from context.

So gradient descent on convex quadratics might have 0, 1, or infinitely many fixed points. If it has fixed points, then gradient descent might or might not converge to one of them. To disentangle these possibilities, we need to analyze the dynamics.

Since gradient descent is invariant to rigid transformations, we're free to analyze the dynamics in a coordinate system where things are simpler. Recall the **Spectral Theorem** from linear algebra: any symmetric matrix $\mathbf{A}$ has a full set of eigenvectors, the corresponding eigenvalues are all real, and the eigenvectors can be taken to be orthogonal. This can be expressed in terms of the **spectral decomposition**:

The Spectral Theorem is probably the most important result from introductory linear algebra, even though it's often omitted from introductory linear algebra courses. If you're not familiar with it, you should take the time now to understand it.

$$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top, \tag{11}$$

where $\mathbf{Q}$ is an orthogonal matrix whose columns contain the eigenvectors of $\mathbf{A}$, and $\mathbf{D}$ is a diagonal matrix whose diagonal entries are the corresponding eigenvalues. Our assumption that $\mathbf{A}$ is positive semidefinite corresponds to all the eigenvalues being nonnegative. What the Spectral Theorem is really saying is that given a symmetric matrix, we can always rotate to a coordinate system where that matrix is diagonal.

Returning to the gradient descent dynamics, let $\mathbf{Q}$ be an orthogonal eigenbasis for $\mathbf{A}$, and apply the transformation $\mathcal{T}(\mathbf{w}) = \mathbf{Q}^\top\mathbf{w}$. By expanding out the transformed cost (Eqn. 8), we obtain a convex quadratic cost function in the new coordinate system:

$$\begin{aligned} \tilde{\mathcal{J}}(\tilde{\mathbf{w}}) &= \tfrac{1}{2}\tilde{\mathbf{w}}^\top\tilde{\mathbf{A}}\tilde{\mathbf{w}} + \tilde{\mathbf{b}}^\top\tilde{\mathbf{w}} + c \\ \tilde{\mathbf{A}} &= \mathbf{Q}^\top\mathbf{A}\mathbf{Q} \\ \tilde{\mathbf{b}} &= \mathbf{Q}^\top\mathbf{b}, \end{aligned} \tag{12}$$

Observe that $\tilde{\mathbf{A}} = \operatorname{diag}(\tilde{a}_1, \ldots, \tilde{a}_D)$ is a diagonal matrix whose entries $\tilde{a}_j$ are the eigenvalues of $\mathbf{A}$. These values represent the **curvature** of the cost function, i.e. they tell us how quickly it curves upwards if you move along the corresponding eigenvector.

The magic of this change-of-basis transformation is that now all the individual coordinates evolve independently. Plugging in the gradient descent update (Eqn. 9), we see that each coordinate independently evolves as:

$$\tilde{w}_j^{(k+1)} \leftarrow \tilde{w}_j^{(k)} - \alpha(\tilde{a}_j\tilde{w}_j^{(k)} + \tilde{b}_j).$$

To analyze the dynamics, we consider 3 separate cases:

Case 1: $\tilde{a}_j > 0$

In this case, the function curves upwards along this eigendirection, and there is a unique fixed point given by

$$\tilde{w}_{\star j} = -\tilde{b}_j/\tilde{a}_j.$$

By rearranging terms, we see that

$$\tilde{w}_j^{(k+1)} - \tilde{w}_{\star j} = (1 - \alpha \tilde{a}_j)(\tilde{w}_j^{(k)} - \tilde{w}_{\star j}).$$

This recurrence has the solution:

$$\tilde{w}_j^{(k)} = \tilde{w}_{\star j} + (1 - \alpha \tilde{a}_j)^k (\tilde{w}_j^{(0)} - \tilde{w}_{\star j}).$$

Now we can break this down into more cases, depending on α and $\tilde{a}_j$:

Case 1(a): $0 < \alpha \tilde{a}_j < 2$

The iterates $\tilde{w}_j^{(k)}$ converge exponentially to $\tilde{w}_{\star j}$, where the rate of convergence is given by $|1 - \alpha \tilde{a}_j|$. If $0 < \alpha \tilde{a}_j < 1$, they approach $\tilde{w}_{\star j}$ monotonically, whereas if $1 < \alpha < 2$, they oscillate as they converge.

Case 1(b): $\alpha \tilde{a}_j = 2$.

The iterates oscillate between $\pm \tilde{w}_j^{(0)}$.

Case 1(c): $\alpha \tilde{a}_j > 2$.

The iterates diverge exponentially.

Case 2: $\tilde{a}_j = 0$ and $\tilde{b}_j \neq 0$.

In this case, the update rule is

$$\tilde{w}_j^{(k+1)} \leftarrow \tilde{w}_j^{(k)} - \alpha \tilde{b}_j,$$

and the recurrence is solved by

$$\tilde{w}_j^{(k)} = \tilde{w}_j^{(0)} - \alpha k \tilde{b}_j.$$

Hence, the weight continues to move linearly and never converges.

Case 3: $\tilde{a}_j = 0$ and $\tilde{b}_j = 0$.

In this case, $\tilde{w}_j$ is never updated, so $\tilde{w}_j^{(k)} = \tilde{w}_j^{(0)}$ for all k.

Clearly, the weights will never converge if any of the dimensions match cases 1(b), 1(c), or 2. In order to get convergence, we must satisfy the following conditions:

A1. $\mathcal{J}$ is bounded below.

A2. $\alpha < 2\tilde{a}_{\max}^{-1}$, where $\tilde{a}_{\max}$ denotes the maximum diagonal entry of $\tilde{\mathbf{A}}$, or equivalently, the maximum eigenvalue of $\mathbf{A}$.

A1 is often easy to verify; e.g., the linear regression cost function is a sum of squares, so it's bounded below by 0. If A1 is satisfied, then this implies $\tilde{b}_j = 0$ whenever $\tilde{a}_j = 0$, so case 2 is ruled out. Assumption A2 rules out cases 1(b) and 1(c). Hence, we're left with only cases 1(a) and 3, implying that coordinates with positive curvature will converge exponentially, and coordinates with zero curvature will remain fixed.

Reasoning about individual coordinates can be tedious, so let's translate this all back into linear algebra. It can be shown that the iterates, in the original coordinate system, are given by:

$$\mathbf{w}^{(k)} = \mathbf{w}^{(\infty)} + (\mathbf{I} - \alpha \mathbf{A})^k (\mathbf{w}^{(0)} - \mathbf{w}^{(\infty)}), \tag{13}$$

The linear algebraic notation is more compact, but be sure you understand how it relates to the coordinatewise analysis.

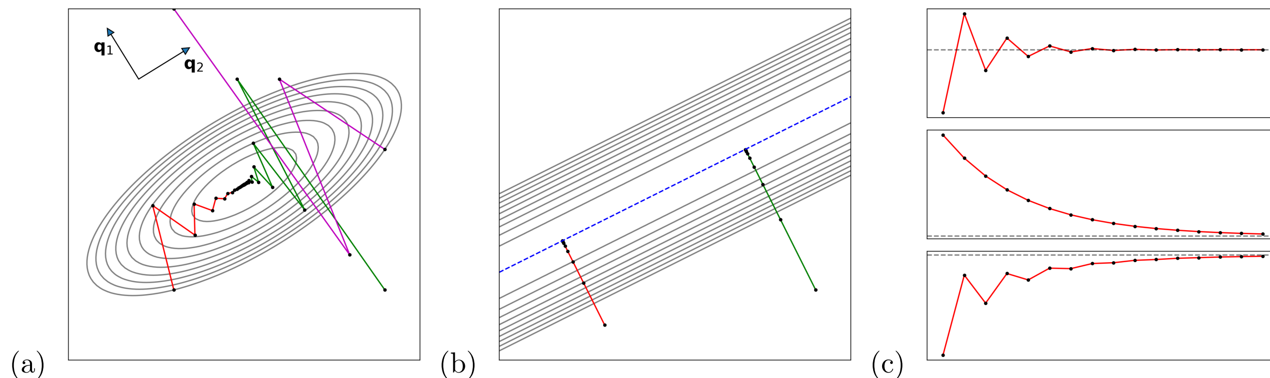

Figure 2: The dynamics of gradient descent on convex quadratics. **(a)** If $\mathbf{A}$ is full rank, and α is small enough, then the iterates converge to the unique global minimum (red and green). If the learning rate is too large, then the iterates diverge (magenta). **(b)** If $\mathbf{A}$ is low rank and α is small enough, then the iterates converge linearly to the closest point in the minimum cost subspace (blue). **(c)** The projections of the iterates from figure (a) onto the eigenvectors $\mathbf{q}_1$ **(top)** and $\mathbf{q}_2$ **(middle)** are exponential functions. The projection onto an arbitrary direction **(bottom)** is generally not an exponential (in fact, it's a superposition of exponentials).

where $\mathbf{w}^{(\infty)}$ is the *projection* of $\mathbf{w}^{(0)}$ onto the *subspace* which achieves the minimum cost:

$$\mathbf{w}^{(\infty)} = \arg\min_{\mathbf{w}} \|\mathbf{w} - \mathbf{w}^{(0)}\|^2 \quad \text{s.t.} \quad \mathbf{w} \in \arg\min_{\mathbf{w}'} \mathcal{J}(\mathbf{w}'),$$

Recall that the minimum cost subspace is given explicitly by:

$$\arg\min_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \{\mathbf{w}_\star + \Delta\mathbf{w} : \Delta\mathbf{w} \in \text{null space of } \mathbf{A}\},$$

where $\mathbf{w}_\star$ is any particular minimizer of $\mathcal{J}$. Observe that the iterates converge exponentially to $\mathbf{w}^{(\infty)}$, which justifies this choice of notation.

To give a convenient equation for $\mathbf{w}^{(\infty)}$, we'll need an operation called the **pseudoinverse**, which is a generalization of the matrix inverse to possibly non-invertible matrices. If a (possibly non-square) matrix $\mathbf{B}$ has the singular value decomposition (SVD) $\mathbf{B} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$, then its pseudoinverse, denoted $\mathbf{B}^\dagger$, is given by:

$$\mathbf{B}^\dagger = \mathbf{V}\mathbf{S}^\dagger\mathbf{U}^\top$$

where the diagonal matrix $\mathbf{S}^\dagger$ has a diagonal entry s_j^{-1} for each nonzero singular value s_j, and an entry of 0 for each 0 singular value. The pseudoinverse can also be written explicitly as

$$\mathbf{B}^\dagger = (\mathbf{B}^\top\mathbf{B})^{-1}\mathbf{B}^\top$$

in the case where $\mathbf{B}^\top\mathbf{B}$ is invertible. Furthermore, it agrees with the ordinary inverse, i.e. $\mathbf{B}^\dagger = \mathbf{B}^{-1}$ if $\mathbf{B}$ is invertible. In general,

$$\mathbf{B}^\dagger\mathbf{c} = \arg\min_{\mathbf{v}} \|\mathbf{v}\|^2 \quad \text{s.t.} \quad \mathbf{v} \in \arg\min_{\mathbf{v}'} \|\mathbf{B}\mathbf{v}' - \mathbf{c}\|^2.$$

For a symmetric matrix $\mathbf{A}$, the SVD agrees with the eigendecomposition $\mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, so the pseudoinverse can be written as:

$$\mathbf{A}^\dagger = \mathbf{Q}\mathbf{D}^\dagger\mathbf{Q}^\top,$$

where $\mathbf{D}^\dagger$ is obtained by replacing diagonal entries with reciprocals or 0, analogously to $\mathbf{S}^\dagger$.

Now back to the minimum norm solution for quadratics. In the case where $\mathbf{w}^{(0)} = \mathbf{0}$ (as is common practice for linear regression), $\mathbf{w}^{(\infty)}$ is the minimum norm vector in the minimum cost subspace. It can be shown that

$$\mathbf{w}^{(\infty)} = -\mathbf{A}^\dagger \mathbf{b}. \tag{14}$$

Exercise: show this. Also, what is the generalization of this formula when $\mathbf{w}^{(0)} \neq \mathbf{0}$?

Note also that if $\mathbf{A}$ is full rank, then the optimal solution $\mathbf{w}^{(\infty)} = \mathbf{w}_\star = -\mathbf{A}^{-1}\mathbf{b}$ is unique (because the null space of $\mathbf{A}$ is trivial).

Now, what happens to the loss? Keeping the transformed coordinate system (and denoting irrelevant constants with c_i), the loss also decomposes into an independent term for each coordinate:

$$\begin{aligned}
\tilde{\mathcal{J}}(\tilde{\mathbf{w}}) &= \tfrac{1}{2}\tilde{\mathbf{w}}^\top \tilde{\mathbf{A}} \tilde{\mathbf{w}} + \tilde{\mathbf{b}}^\top \tilde{\mathbf{w}} + c_1 \\
&= \sum_j \left[\frac{\tilde{a}_j}{2} \tilde{w}_j^2 + \tilde{b}_j \tilde{w}_j \right] + c_1 \\
&= \sum_{j : \tilde{a}_j > 0} \frac{\tilde{a}_j}{2} (\tilde{w}_j - \tilde{w}_{\star j})^2 + \sum_{j : \tilde{a}_j = 0} \tilde{b}_j \tilde{w}_j + c_2.
\end{aligned}$$

The first term corresponds to Case 1, while the second term corresponds to Cases 2 and 3. But note that in Case 3, $\mathbf{b}_j = 0$, so Case 3 doesn't contribute to the objective. As long as A1 is satisfied, Case 2 is impossible, so the second term drops out entirely. This leaves Case 1. Plugging in our formula for $\tilde{w}_j^{(k)}$, we get:

Observe that in Case 2, the objective is unbounded below, so the optimum is $-\infty$. This obviously can't happen for linear regression, since the objective is a sum of squares and therefore nonnegative.

$$\mathcal{J}(\mathbf{w}^{(k)}) = \tilde{\mathcal{J}}(\tilde{\mathbf{w}}^{(k)}) = \sum_{j : \tilde{a}_j > 0} \frac{\tilde{a}_j}{2} (\tilde{w}_j^{(0)} - \tilde{w}_{\star j})^2 (1 - \alpha \tilde{a}_j)^{2k}.$$

If condition A2 isn't satisfied, then some of the terms will grow exponentially — not too surprising, since we know the iterates diverge in that case. As long as A2 is satisfied, each term decays exponentially as $(1 - \alpha\tilde{a}_j)^{2k}$. Observe that for small $\alpha \tilde{a}_j$, we have

$$(1 - \alpha\tilde{a}_j)^{2k} \approx \exp(-2\alpha\tilde{a}_j k). \tag{15}$$

In the field of optimization, this exponential decay is referred to as **linear convergence**, to contrast it with the quadratic convergence obtained by second-order optimizers. I'll avoid using this term since it's confusing in the context of ML, where we often want to contrast exponential convergence with sub-exopnential convergence rates we find in the stochastic optimization setting.

Hence, $2\alpha\tilde{a}_j$ can be thought of as the rate of convergence for that dimension.

At the beginning of training, it's likely that many of the high curvature eigenvalues will contribute significantly to the cost, hence the cost will appear to decay at a rate faster than $2\alpha\tilde{a}_{\min}$. However, the term(s) which decay the most slowly, i.e. the lowest curvature dimension(s), will eventually come to dominate. Hence, asymptotically, the loss will decay proportionally to $(1 - \alpha\tilde{a}_{\min})^2$. (We use $\tilde{a}_{\min}$ to denote the minimum nonzero eigenvalue.) This behavior is shown in Figure 3.

Since the cost is asymptotically dominated by the lowest curvature di-

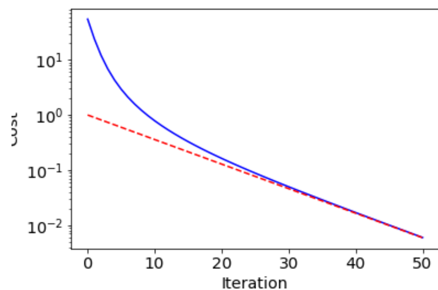

Figure 3: Convergence of gradient descent on a quadratic objective with curvature eigenvalues $\{1, \ldots, 10\}$. Blue is the total cost, while red is the cost only for the lowest curvature direction. Note the log scale on the y-axis.

rection(s), we can model the asymptotic behavior as follows:

$$\begin{aligned}
\mathcal{J}(\mathbf{w}^{(k)}) - \mathcal{J}(\mathbf{w}_\star) &\approx \mathcal{J}_{\text{min}}(\mathbf{w}^{(k)}) && \text{for large } k \\
&= (1 - \alpha \tilde{a}_{\text{min}})^{2k} \mathcal{J}_{\text{min}}(\mathbf{w}^{(0)}) \\
&\approx \exp(-2\alpha \tilde{a}_{\text{min}} k)\, \mathcal{J}_{\text{min}}(\mathbf{w}^{(0)}) && \text{Eqn. 15} \\
&\approx \exp\left(-\frac{4\tilde{a}_{\text{min}}}{\tilde{a}_{\text{max}}} k\right) \mathcal{J}_{\text{min}}(\mathbf{w}^{(0)}) && \text{Assumption A1} \\
&= \exp(-4\kappa^{-1} k)\, \mathcal{J}_{\text{min}}(\mathbf{w}^{(0)}).
\end{aligned}$$

Here, $\mathcal{J}_{\text{min}}$ denotes the component(s) of the cost corresponding to the minimum curvature direction(s), and $\kappa = \tilde{a}_{\text{min}}^{-1} \tilde{a}_{\text{max}}$ is known as the **condition number**. In general, κ measures the difficulty of the optimization problem, and analogues can be defined for non-quadratic objectives as well. The smallest possible condition number is $\kappa = 1$, and if κ is close to 1, the problem is said to be **well-conditioned**. If $\kappa \gg 1$, the problem is said to be **ill-conditioned**.

Another way to think about the above convergence rate is that if you want to reach some small cost ϵ, it will require $\mathcal{O}(\kappa \log \frac{1}{\epsilon})$ iterations. Hence, gradient descent is said to be an $\mathcal{O}(\kappa)$ algorithm. (Here, we ignore the dependence on ϵ.) We'll soon see that it's possible to achieve $\mathcal{O}(\sqrt{\kappa})$ for convex quadratics using other gradient-based iterative algorithms.

2.3 Without Loss of Generality

In the above derivation, we explicitly **reparameterized** the problem, i.e. transformed it to a coordinate system where $\mathbf{A}$ was diagonal. While this works, it results in a clunky notation with lots of tildes. We'd like to avoid clunky notation, if possible. So instead, we will say something like, "Assume we choose a coordinate system such that $\mathbf{A}$ is diagonal", or more tersely, "Assume **without loss of generality (WLOG)** that $\mathbf{A}$ is diagonal." The meaning of WLOG generally needs to be inferred from context, but what it means here is that it is always possible to transform the problem to a coordinate system which satisfies some particular property. In the above

derivation, we exploited the rotation invariance of gradient descent, combined with the Spectral Theorem.

When making a WLOG argument, it's necessary to keep in mind what constraints were already imposed on the coordinate system. E.g., if one already assumed WLOG that $\mathbf{A}$ is diagonal, then one can't also assume WLOG that some other symmetric matrix $\mathbf{B}$ is also diagonal, since there might not be any coordinate system that simultaneously diagonalizes both matrices. (Unless, of course, $\mathbf{A}$ and $\mathbf{B}$ are known to be codiagonalizable.) It's also important to ensure that the dynamics are actually rotation invariant. E.g., one can't assume diagonal $\mathbf{A}$ WLOG when analyzing an optimization algorithm based on diagonal rescaling, such as RMSprop or Adam.

This will all come naturally once you're used to it. But if you're confused about whether WLOG is justified, you need to be able to go through the same sort of analysis we did above.

When analyzing the behavior of an update rule close to an optimum or a fixed point $\mathbf{w}_\star$, it is common to assume WLOG that $\mathbf{w}_\star = \mathbf{0}$. What this means is that, as long as the dynamics are translation invariant (as is normally the case), we can transform to a coordinate system whose origin is at $\mathbf{w}_\star$.

2.4 Back to Linear Regression

Now let's apply what we've learned to linear regression. Recall that we're trying to minimize

$$\begin{aligned}
\mathcal{J}(\breve{\mathbf{w}}) &= \tfrac{1}{2N}\|\breve{\boldsymbol{\Phi}}\breve{\mathbf{w}} - \mathbf{t}\|^2 \\
&= \tfrac{1}{2N}\breve{\mathbf{w}}^\top \breve{\boldsymbol{\Phi}}^\top \breve{\boldsymbol{\Phi}}\breve{\mathbf{w}} - \tfrac{1}{N}\mathbf{t}^\top \breve{\boldsymbol{\Phi}}\breve{\mathbf{w}} + \tfrac{1}{2N}\mathbf{t}^\top \mathbf{t}.
\end{aligned}$$

Clearly this is an instance of the optimization problem in Eqn. 4 with $\mathbf{A} = \frac{1}{N}\breve{\boldsymbol{\Phi}}^\top \breve{\boldsymbol{\Phi}}$ and $\mathbf{b} = -\frac{1}{N}\breve{\boldsymbol{\Phi}}^\top \mathbf{t}$. (Recall that the $\breve{\cdot}$ notation indicates the homogeneous parameterization.)

It's easy to check that condition A1 is satisfied, since the cost function is a sum of squares and therefore nonnegative. So assuming a small enough learning rate (as in A2), the iterates are given by Eqn. 13:

$$\breve{\mathbf{w}}^{(k)} = \breve{\mathbf{w}}^{(\infty)} + (\mathbf{I} - \alpha\mathbf{A})^k (\breve{\mathbf{w}}^{(0)} - \breve{\mathbf{w}}^{(\infty)}),$$

where $\breve{\mathbf{w}}^{(\infty)}$ is, among the set of all optimal weight vectors, the one *closest* to the initialization $\breve{\mathbf{w}}^{(0)}$. In particular, if $\breve{\mathbf{w}}^{(0)} = \mathbf{0}$ (as is common practice for linear regression), then $\breve{\mathbf{w}}^{(\infty)}$ is the minimum norm solution. We saw earlier (see Eqn. 14) that the min-norm solution is given explicitly by

The last step uses the identity $(\mathbf{B}^\top\mathbf{B})^\dagger \mathbf{B}^\top = \mathbf{B}^\dagger$, which holds for any matrix $\mathbf{B}$.

$$\begin{aligned}
\breve{\mathbf{w}}^{(\infty)} &= -\mathbf{A}^\dagger \mathbf{b} \\
&= (\breve{\boldsymbol{\Phi}}^\top \breve{\boldsymbol{\Phi}})^\dagger \breve{\boldsymbol{\Phi}}^\top \mathbf{t} \qquad (16) \\
&= \breve{\boldsymbol{\Phi}}^\dagger \mathbf{t}.
\end{aligned}$$

It's interesting to compare this result with the optimal solution to **ridge regression**, i.e. linear regression with an ℓ_2 regularization term. Recall that this involves penalizing the norm of the weights:

Note that, in practice, we don't normally regularize the bias parameter, so this formulation isn't quite standard.

$$\mathcal{J}_\lambda(\breve{\mathbf{w}}) = \tfrac{1}{2N}\|\breve{\boldsymbol{\Phi}}\breve{\mathbf{w}} - \mathbf{t}\|^2 + \frac{\lambda}{2}\|\breve{\mathbf{w}}\|^2,$$

where $\lambda > 0$ is a hyperparameter controlling the strength of the regularization. This is also a quadratic cost function, with $\mathbf{A} = \frac{1}{N}\breve{\boldsymbol{\Phi}}^\top \breve{\boldsymbol{\Phi}} + \lambda \mathbf{I}$ and $\mathbf{b} = \frac{1}{N}\breve{\boldsymbol{\Phi}}^\top \mathbf{t}$. Unlike the unregularized case, $\mathbf{A}$ is guaranteed to be positive definite, because $\breve{\boldsymbol{\Phi}}^\top \breve{\boldsymbol{\Phi}}$ is positive semidefinite and $\lambda \mathbf{I}$ is positive definite. Therefore, there is always a unique optimal solution

$$\breve{\mathbf{w}}_\lambda = \arg\min_{\breve{\mathbf{w}}} \mathcal{J}_\lambda(\breve{\mathbf{w}}) = (\breve{\boldsymbol{\Phi}}^\top \breve{\boldsymbol{\Phi}} + \lambda \mathbf{I})^{-1} \breve{\boldsymbol{\Phi}}^\top \mathbf{t}.$$

Larger values of λ imply stronger regularization, and $\lambda = 0$ turns off the regularization entirely. We already saw that $\mathbf{w}_\lambda$ is undefined for $\lambda = 0$ (since the optimum might not be unique), but we can instead take the limit as $\lambda \to 0$:

$$\lim_{\lambda \to 0} \breve{\mathbf{w}}_\lambda = \breve{\boldsymbol{\Phi}}^\dagger \mathbf{t}.$$

You can determine the limit by expanding out the SVD of $\breve{\boldsymbol{\Phi}}$ and simplifying.

This coincides exactly with the stationary solution $\breve{\mathbf{w}}^{(\infty)}$ to the unregularized problem! This phenomenon, where the dynamics of optimization produce regularization-like behavior even in the absence of an explicit regularizer, is known as **implicit regularization**. This is an important topic of study in deep learning today, and will be a recurring theme throughout the course.

3 Why Normalize the Features?

A common preprocessing trick in deep learning, and machine learning more generally, is to **normalize**, or **standardize**, the input features by transforming them to have zero mean and unit variance. Mathematically,

$$\begin{aligned}
\tilde{\phi}_j(\mathbf{x}) &= \frac{\phi_j(\mathbf{x}) - \mu_j}{\sigma_j} \\
\mu_j &= \frac{1}{N} \sum_{i=1}^N \phi_j(\mathbf{x}^{(i)}) \\
\sigma_j^2 &= \frac{1}{N} \sum_{i=1}^N (\phi_j(\mathbf{x}^{(i)}) - \mu_j)^2
\end{aligned}$$

Observe that the transformed features have zero mean and unit variance. Why is this a good idea?

Sometimes we transform the data to be zero mean, but skip the rescaling step. This is known as **centering**, and data with zero mean is said to be **centered**.

Intuitively, many features have arbitrary units. Lengths can be measured in feet or miles, and time can be measured in seconds or years. The choice of zero can also be arbitrary, for instance if the input represents a calendar year. Clearly, we don't want our statistical analysis to depend on an arbitrary choice of units, and normalizing the inputs eliminates this unwanted source of variability. But what actually goes wrong mechanistically if we don't normalize?

In some cases, nothing goes wrong. For instance, suppose we fit an unregularized linear regression model, collect enough data to ensure a unique optimal solution, and compute the exact optimal solution. In this case, we find the optimal linear predictor of the targets from the input features,

and (as a basic fact of linear algebra) this predictor is invariant to linear reparameterizations of the features (of which normalization is an instance). Hence, normalization doesn't matter.

However, the dynamics of gradient descent are *not* invariant to linear transformations of the inputs, and neither is the minimum norm solution $\breve{\mathbf{w}}^{(\infty)}$ that the iterates converge to. To understand the gradient descent dynamics, consider the curvature matrix $\mathbf{A} = \frac{1}{N}\breve{\mathbf{\Phi}}^\top \breve{\mathbf{\Phi}}$:

$$\begin{aligned}
\mathbf{A} &= \tfrac{1}{N}\breve{\mathbf{\Phi}}^\top \breve{\mathbf{\Phi}} \\
&= \tfrac{1}{N}\begin{pmatrix} \mathbf{\Phi}^\top \\ \mathbf{1}^\top \end{pmatrix} \begin{pmatrix} \mathbf{\Phi} & \mathbf{1} \end{pmatrix} \\
&= \begin{pmatrix} \frac{1}{N}\sum_{i=1}^N \boldsymbol{\phi}(\mathbf{x}^{(i)})\boldsymbol{\phi}(\mathbf{x}^{(i)})^\top & \frac{1}{N}\sum_{i=1}^N \boldsymbol{\phi}(\mathbf{x}^{(i)}) \\ \frac{1}{N}\sum_{i=1}^N \boldsymbol{\phi}(\mathbf{x}^{(i)})^\top & 1 \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^\top & \boldsymbol{\mu} \\ \boldsymbol{\mu}^\top & 1 \end{pmatrix}
\end{aligned} \tag{17}$$

Here, $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ denote the empirical feature mean and covariance, respectively. The final step uses the identity $\mathbb{E}[\mathbf{v}\mathbf{v}^\top] = \mathrm{Cov}(\mathbf{v}) + \mathbb{E}[\mathbf{v}]\mathbb{E}[\mathbf{v}]^\top$, which holds for any random vector $\mathbf{v}$.

Recall that, to understand the dynamics, we are interested in the eigenvectors and eigenvalues of $\mathbf{A}$. It's hard to say much about the general case that provides much insight, so instead we'll try to get some intuition by plugging in in a few particular choices of $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$.

1. Suppose $\boldsymbol{\mu} = \mathbf{0}$ and $\mathbf{\Sigma} = \mathbf{I}$. Such a data distribution is referred to as **white**, by analogy with white noise. Plugging these values into Eqn. 17, you can check that $\mathbf{A} = \mathbf{I}$, the $(D+1) \times (D+1)$ identity matrix. This is the ideal case for optimization, because the condition number is $\kappa = 1$. Gradient descent heads directly for the optimal solution.

2. Suppose $\boldsymbol{\mu} = \mathbf{0}$ and $\mathbf{\Sigma}$ is a diagonal matrix whose entries are in the interval $[\sigma^2_{\mathrm{min}}, \sigma^2_{\mathrm{max}}]$, with $\sigma^2_{\mathrm{min}} \leq 1 \leq \sigma^2_{\mathrm{max}}$. The data are centered, but not white. Then $\mathbf{A}$ is also a diagonal matrix, whose diagonal entries are the feature variances (i.e. the diagonal entries of $\mathbf{\Sigma}$), plus a 1 for the homogeneous coordinate. The coordinates which train the fastest (i.e. the ones with the highest curvature) are the features with the largest variance. The smaller variance dimensions train more slowly. The condition number is $\kappa = \sigma^2_{\mathrm{max}}/\sigma^2_{\mathrm{min}}$, so it is beneficial for the speed of optimization that the variances of different dimensions be close together.

3. As before, suppose $\mathbf{\Sigma}$ is a diagonal matrix whose entries are in the interval $[\sigma^2_{\mathrm{min}}, \sigma^2_{\mathrm{max}}]$, but now let $\boldsymbol{\mu} \neq \mathbf{0}$. Hence, the data are uncentered.

   We start by decomposing Eqn. 17:

$$\mathbf{A} = \begin{pmatrix} \mathbf{\Sigma} & \\ & 0 \end{pmatrix} + \begin{pmatrix} \boldsymbol{\mu} \\ 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{\mu}^\top & 1 \end{pmatrix}$$

Let's look at the two terms separately. The first term is a diagonal matrix whose nonzero eigenvalues are the diagonal entries of $\mathbf{\Sigma}$, which are bounded between σ^2_{min} and σ^2_{max}. The second term, as stated above, is a rank-1 matrix whose nonzero eigenvalue is $\|\boldsymbol{\mu}\| + 1$. As we increase the feature dimension D, the eigenvalues of the first term remain bounded, while the nonzero eigenvalue of the second term grows proportionally to D.

In general, there is no direct relationship between the eigendecomposition of a sum $\mathbf{C} + \mathbf{D}$ and the eigendecomposition of the component matrices $\mathbf{C}$ and $\mathbf{D}$. However, in the present case it is possible to show the following: if the nonzero eigenvalue of the second term is much larger than the eigenvalues of the first term, then $\mathbf{A}$ will have a large outlier eigenvalue greater than or equal to $\|\boldsymbol{\mu}\| + 1$, approximately in the direction $(\boldsymbol{\mu}^\top \; 1)^\top$. The remaining eigenvalues will be in the range $[0, \sigma^2_{\text{max}}]$.

I don't know of an easy derivation of this fact about eigenvalues. It can be shown using the variational characterization of eigenvalues; see, e.g., Section 4.3 of Horn and Johnson (2012).

Hence, the condition number is bounded as $\kappa \geq (\|\boldsymbol{\mu}\|^2 + 1)/\sigma^2_{\text{max}}$, which grows linearly in the feature dimension. We can actually say a bit more. The maximum stable learning rate is bounded by $(\|\boldsymbol{\mu}\|^2 + 1)^{-1}$, while most of the signal is contained in directions whose curvature is bounded above by σ^2_{max}. Hence, as the dimension increases, the learning rate must get smaller, and hence the signal will be learned more and more slowly. This problem is solved if we center the data, as then $\boldsymbol{\mu} = \mathbf{0}$.

So we've seen examples of how uncentered and unnormalized data can make optimization more difficult by increasing the condition number. Hence, it should be intuitive that normalizing the data will improve the conditioning.

3.1 Full Whitening

Sometimes we go even further and **whiten** the data, by transforming it to have zero mean and unit covariance. This can be achieved with the transformation

$$\tilde{\boldsymbol{\phi}}(\mathbf{x}) = \mathbf{S}^{-1}(\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\mu}),$$

where $\mathbf{S}$ is any matrix such that $\mathbf{S}\mathbf{S}^\top = \mathbf{\Sigma}$. Based on example 1 above, this is clearly the optimal choice from the standpoint of optimization, i.e. if we are simply trying to minimize the training loss as fast as possible. But we need to be careful: if the data covariance contains useful information, then whitening might hurt generalization.

Possible choices for $\mathbf{S}$ include (1) the matrix square root $\mathbf{\Sigma}^{1/2} = \mathbf{Q}\mathbf{D}^{1/2}\mathbf{Q}$, where $\mathbf{\Sigma} = \mathbf{Q}\mathbf{D}\mathbf{Q}$ is the spectral decomposition; or (2) the Cholesky factor $\mathbf{L}$, i.e. the unique lower triangular matrix such that $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^\top$.

In general, gradient descent will train faster in the directions of larger feature variance, and the stationary solution will favor explaining the data using the directions of higher variance. Whether this is a feature or a bug depends on the context. When we discussed centering and coordinatewise rescaling, we observed that the mean and variance were determined by arbitrary choices of units. Since this source of variability contains no useful information, we might as well get rid of it.

However, the *correlations* between different dimensions might contain useful information. It's a common assumption that more highly correlated features are more likely to contain signal rather than noise; this is

the motivation for the common step of preprocessing the data by projecting onto the dominant principal components. Hence, the fact that gradient descent learns more quickly in the top principal components can be seen as a kind of **inductive bias**, or modeling assumption, and can help generalization. Hence, despite the optimization benefits, we can't always assume that whitening is a good idea. In practice, it is rarely done when training neural nets, even when it's computationally tractable.

Indeed, sometimes even coordinatewise rescaling can be unhelpful, since the variances can contain information about the relative importance of different input dimensions. E.g., in MNIST, the boundary pixels have small variance because they contain no signal, so we don't want to amplify them.

As an example of this phenomenon, consider a toy problem where each data instance is generated by first sampling a scalar-valued target $t \sim \mathcal{N}(0, 1)$, and the inputs consist of independent noisy observations of t, i.e. each $x_j \sim \mathcal{N}(t, \sigma_x^2)$ for $\sigma_x^2 \ll 1$. We observe the input vector $\mathbf{x}$ and a noisy version of the target, $\hat{t} \sim \mathcal{N}(t, \sigma_t^2)$. You can check that the covariance $\boldsymbol{\Sigma}$ contains one large eigenvalue corresponding to the signal, and the remaining eigenvalues are much smaller and correspond to noise. Gradient descent, therefore, will quickly learn the signal and only later overfit by fitting the noise. By whitening the data, we can dramatically speed up optimization of the training loss, based on the arguments above. However, we achieve this by speeding up convergence along the noise directions, resulting in severe overfitting. We can fix this overfitting by collecting lots more data or perhaps applying a clever regularizer, but at the end of the day we still won't get any speedup on the validation set because gradient descent was already fitting the signal essentially as fast as possible.

This dilemma exemplifies a recurring theme in this course: we can't fully decouple optimization from generalization. If we discover a trick that speeds up the optimization of the training loss on a finite dataset, we shouldn't blindly apply it. We need to think carefully about *why* the trick speeds up optimization, and then think about whether this effect will speed up *generalization* as well. If the trick speeds up learning of the true underlying structure, then it might help us achieve low validation error faster (or sometimes even converge to a lower asymptotic validation error). On the other hand, if the trick speeds up training set convergence by more efficiently memorizing training examples, then this won't translate into improvements on the validation set no matter how carefully we regularize. We'll see lots more examples of this as the course goes on.

3.2 Lessons for Neural Nets

The effects of normalization have long been understood by neural net researchers; in fact, the analysis above is loosely based on LeCun (1998)'s classic tutorial, "Efficient Backprop". This sort of analysis was the source of various classic tricks for neural net training, including (most directly) normalizing the inputs.

Impressively, most of the advice from Efficient Backprop is still relevant today. It is also very prescient, in that it discussed or touched upon many of the key ideas from the first half of this course.

Less obviously, the scaling and centering of intermediate activations affect optimization in much the same ways as scaling and centering of the inputs, but are harder to control. (We'll make this claim more precise in a later lecture.) Back before the days of ReLU activations, the vanilla activation function was the logistic function:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

The problem is, the outputs range from 0 to 1, meaning the activations must have nonzero mean. The clever solution (which was standard practice for neural net training pre-ReLU) was to replace the activation function with tanh:

$$\tanh(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}.$$

Interestingly, ReLU suffers from the same uncentering problem as the logistic function, but no solution analogous to tanh has been adopted. Batch norm was invented shortly after ReLU became popular, so evidently it was timed just right to meet the newfound need for activation centering!

It can be shown that tanh can be obtained from the logistic function through affine transformations of inputs and outputs, implying that networks with logistic and tanh activations are equally expressive. But the tanh function ranges from -1 to 1, so its outputs aren't *necessarily* uncentered. Therefore, tanh networks can in practice train much faster than logistic networks. Based on this line of reasoning, there were various attempts to find ways to exactly center (and sometimes rescale) the activations. The first one to become widely adopted was batch norm (discussed in detail in Chapter 5).

4 Double Descent

We introduced the phenomenon of "double descent" in the Introduction. This phenomenon happens for plain (unregularized) linear regression as well as neural nets. Ideally, we'd like to use a linear regression model to understand why increasing model capacity past the interpolation threshold can improve generalization. Unfortunately, there isn't a clean analogue of "increasing the number of parameters" of a linear regression model, since the number of parameters is the same as the input dimension D, and adding input dimensions makes more information available to the learner (unlike adding more hidden units). Instead, let's look at the behavior for fixed D as the number of data points N is varied. We set $D = 50$, sample the input dimensions i.i.d. $x_j \sim \mathcal{N}(0, 1)$, compute the true labels as $t = \mathbf{w}^\top \mathbf{x}$ for $\mathbf{w} \sim \mathcal{N}(0, 0.1)$, and observe noisy versions of the targets as $\hat{t} \sim \mathcal{N}(t, 1)$. The training set consists of $\{(\mathbf{x}^{(i)}, \hat{t}^{(i)})\}_{i=1}^N$. We use the raw inputs (i.e. $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$) and we exactly compute the stationary (i.e. min-norm) weights $\mathbf{w}^{(\infty)}$ for the training set. Figure 4(a) shows the training and test error as N is varied from 1 to 200. We observe a double descent effect, whereby the test error peaks at $N = D$, a point called the **interpolation threshold**. This is perhaps even more surprising than the original double descent effect (where the number of parameters was varied), since we wouldn't expect *adding more data* to hurt the performance.

Hastie et al. (2019) analyze the limit for random linear regression problems as $D \to \infty$, $N \to \infty$, and $D/N \to \gamma$. Analyzing the asymptotic risk as γ is varied gives a more direct analogue of the double descent effect for neural nets. But this construction is a bit too involved for an introductory lecture.

The term *interpolation threshold* refers to the idea that when $D > N$, the model is basically interpolating the training data. The regime where $D > N$ is known as the *interpolation regime.*

What's going on? For $N > D$, with high probability the matrix $\mathbf{X}^\top \mathbf{X}$ is invertible, so $\mathbf{w}^{(\infty)}$ is the unique global minimum of the cost function. As $N \to \infty$, the test error decreases because there's more information and less overfitting, just as classical learning theory predicts. But when $N < D$, the model is **overparameterized**, so $\mathbf{w}^{(\infty)}$ is the minimum norm solution. Intuitively, when $N \approx D$, it's just barely possible to fit all the training data, i.e. this requires a large norm for $\mathbf{w}^{(\infty)}$. When N is much smaller than D, it's possible to fit the training data using a much smaller weight norm. Smaller norm weight vectors are more robust to overfitting (hence the motivation for ℓ_2 regularization), so they might generalize better.

To understand this mathematically, consider the closed-form expression for the min-norm weights, $\mathbf{w}^{(\infty)} = \mathbf{X}^\dagger \mathbf{t}$ (see Eqn. 16). Roughly speaking,

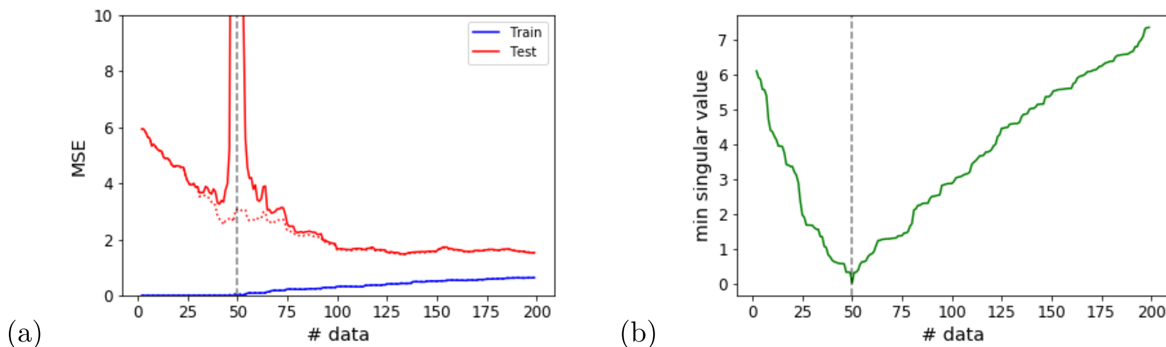

Figure 4: Double descent effect for linear regression. **(a)** Training and test error as a function of N, the number of training examples. The solid line is the unregularized model, and the dotted line uses ℓ_2 regularization with $\lambda = 1$. (The regularized and unregularized training set curves are nearly indistinguishable.) The dashed line indicates the interpolation threshold, $N = D$. **(b)** The minimum nonzero singular value of $\mathbf{X}$, as a function of N.

$\mathbf{w}^{(\infty)}$ will be large when $\mathbf{X}^\dagger$ is large. Since $\mathbf{X}^\dagger$ is defined by inverting the nonzero singular values of $\mathbf{X}$, it will be large when $\mathbf{X}$ has small nonzero singular values. Figure 4(b) plots the smallest nonzero singular value of $\mathbf{X}$ as a function of D. The minimum singular value gets closest to zero at the interpolation threshold. Explaining why this happens is beyond the scope of this course, but it follows from a basic result of a field of mathematics called random matrix theory.

The observation that performance can get worse as more data is added indicates that the double descent effect is somehow pathological. Adding more information shouldn't hurt. Indeed, the pathology can be basically eliminated (in our regression example) by adding a small amount of ℓ_2 regularization ($\lambda = 1$), without noticeably hurting the training or test error outside the pathological regime, as shown in Figure 4(b). The fact that double descent still happens even for modern, well-tuned neural nets is a reflection of how we haven't found a regularizer for neural nets that's as principled and robust as ℓ_2 regularization is for linear regression.

See Nakkiran et al. (2019) for lots of empirical demonstrations of various forms of double descent in modern neural nets.

The above discussion only scratches the surface of what's known about double descent. Hastie et al. (2019) gave basically a complete characterization of double descent for linear regression. They broke the generalization risk down into bias and variance terms, and determined the asymptotic behavior of these terms under a variety of assumptions about the data distribution. It all boils down to random matrix theory.

5 Beyond Linear Regression: Strongly Convex Optimization

We've done a lot of work analyzing one particular system: gradient descent for linear regression. What about other cost functions? Gradient descent dynamics are hard to analyze in general, since a lot of different things can happen. General cost functions can have saddle points and local optima.

Even for convex objectives, gradient descent is a discrete time system, and so could have pretty complex behavior. So in order to prove convergence, we'll need to make some assumptions.

When analyzing optimization algorithms, it's common to assume **strong convexity** — that the cost function curves upwards reasonably quickly. It's also common to make some sort of smoothness assumption so that the gradient doesn't change too quickly. Here are a fairly representative set of assumptions. They are satisfied for various problems we might be interested in solving in machine learning, such as ridge regression or ℓ_2-regularized logistic regression.

Don't worry about understanding this section in detail. We won't build on it very much; I just include it to give you a sense of what a proper optimization proof looks like.

C1. Strong convexity. For any two points $\mathbf{w}$ and $\mathbf{w}'$,

$$\mathcal{J}(\mathbf{w}') \geq \mathcal{J}(\mathbf{w}) + \nabla \mathcal{J}(\mathbf{w})^\top (\mathbf{w}' - \mathbf{w}) + \frac{m}{2} \|\mathbf{w}' - \mathbf{w}\|^2.$$

C2. Lipschitz smoothness. For any two points $\mathbf{w}$ and $\mathbf{w}'$,

$$\|\nabla \mathcal{J}(\mathbf{w}') - \nabla \mathcal{J}(\mathbf{w})\| \leq M \|\mathbf{w}' - \mathbf{w}\|.$$

Under these assumptions, we can show the following:

Theorem 1. *For a differentiable cost function $\mathcal{J}$ satisfying assumptions C1 and C2, if the learning rate $\alpha \leq \frac{1}{M}$, using the gradient descent update*

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha \nabla \mathcal{J}(\mathbf{w}^{(k)}),$$

the cost converges as:

$$\mathcal{J}(\mathbf{w}^{(k+1)}) - \mathcal{J}(\mathbf{w}_\star) \leq (1 - \alpha m)(\mathcal{J}(\mathbf{w}^{(k)}) - \mathcal{J}(\mathbf{w}_\star)).$$

Proof. The proof has two steps. First, we need to show that each update reduces the cost by a sufficient amount. Second, we need to show that the optimal cost cannot be very much better than the current one.

To show a sufficient decrease, we apply the Fundamental Theorem of Calculus to $\mathcal{J}(\mathbf{w}^{(k+1)}) - \mathcal{J}(\mathbf{w}^{(k)})$. The idea is to use Lipschitz smoothness (C2) to show that the trajectory continues to move downhill at a certain rate, because we start out opposite the gradient direction and the gradient doesn't change too much. Representing the interpolation between two consecutive iterates as

$$\mathbf{w}^{(k+\lambda)} = \lambda \mathbf{w}^{(k+1)} + (1 - \lambda)\mathbf{w}^{(k)} = \lambda \alpha \nabla \mathcal{J}(\mathbf{w}^{(k)}),$$

we have:

$$\begin{aligned} \mathcal{J}(\mathbf{w}^{(k+1)}) - \mathcal{J}(\mathbf{w}^{(k)}) &= \int_0^1 (\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)})^\top \nabla \mathcal{J}(\mathbf{w}^{(k+\lambda)})\, \mathrm{d}\lambda \\ &= \underbrace{\int_0^1 (\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)})^\top \nabla \mathcal{J}(\mathbf{w}^{(k)})\, \mathrm{d}\lambda}_{=\mathcal{T}_1} + \underbrace{\int_0^1 (\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)})^\top \left[\nabla \mathcal{J}(\mathbf{w}^{(k+\lambda)}) - \nabla \mathcal{J}(\mathbf{w}^{(k)})\right] \mathrm{d}\lambda}_{=\mathcal{T}_2} \end{aligned}$$

Plugging in the update rule, we get that $\mathcal{T}_1 = -\alpha \|\nabla \mathcal{J}(\mathbf{w}^{(k)})\|^2$. On the other hand,

$$
\begin{aligned}
\mathcal{T}_2 &\leq \int_0^1 \|\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}\| \cdot \|\nabla \mathcal{J}(\mathbf{w}^{(k+\lambda)}) - \nabla \mathcal{J}(\mathbf{w}^{(k)})\| \, \mathrm{d}\lambda && \text{(Cauchy-Schwartz)} \\
&\leq \int_0^1 \|\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}\| \cdot M \|\mathbf{w}^{(k+\lambda)} - \mathbf{w}^{(k)}\| \, \mathrm{d}\lambda && \text{(C2)} \\
&= \int_0^1 \alpha^2 M \lambda \|\nabla \mathcal{J}(\mathbf{w}^{(k)})\|^2 \, \mathrm{d}\lambda \\
&= \frac{\alpha^2 M}{2} \|\nabla \mathcal{J}(\mathbf{w}^{(k)})\|^2 \\
&\leq \frac{\alpha}{2} \|\nabla \mathcal{J}(\mathbf{w}^{(k)})\|^2 && \text{(assumption on } \alpha)
\end{aligned}
$$

Putting these results together,

$$
\mathcal{J}(\mathbf{w}^{(k+1)}) - \mathcal{J}(\mathbf{w}^{(k)}) = \mathcal{T}_1 + \mathcal{T}_2 \leq -\frac{\alpha}{2} \|\nabla \mathcal{J}(\mathbf{w}^{(k)})\|^2. \tag{18}
$$

The second part of the proof is to show that the current point is not too suboptimal. Intuitively, this follows directly from strong convexity (C1): this assumption lower bounds the cost function with a quadratic, so $\mathcal{J}(\mathbf{w}_\star)$ can be no smaller than the minimum of this quadratic:

$$
\begin{aligned}
\mathcal{J}(\mathbf{w}_\star) &\geq \mathcal{J}(\mathbf{w}^{(k)}) + \nabla \mathcal{J}(\mathbf{w}^{(k)})^\top (\mathbf{w}_\star - \mathbf{w}^{(k)}) + \frac{m}{2} \|\mathbf{w}_\star - \mathbf{w}^{(k)}\|^2 \\
&\geq \mathcal{J}(\mathbf{w}^{(k)}) - \frac{1}{2m} \|\nabla \mathcal{J}(\mathbf{w}^{(k)})\|^2. \qquad (19)
\end{aligned}
$$

Combining Equations 18 and 19, we find that

$$
\mathcal{J}(\mathbf{w}^{(k+1)}) - \mathcal{J}(\mathbf{w}_\star) \leq (1 - \alpha m)(\mathcal{J}(\mathbf{w}^{(k)}) - \mathcal{J}(\mathbf{w}_\star)).
$$

□

We can relate this result to our analysis of convex quadratics. For a quadratic cost function, C1 lower bounds the minimum eigenvalue of $\mathbf{A}$ as m. Similarly, C2 upper bounds the maximum eigenvalue as M. Analogously to quadratics, the learning rate is bounded by $\frac{1}{M}$. Assuming we set $\alpha = \frac{1}{M}$, the cost decreases in each iteration by a factor of $1 - \frac{m}{M}$. The quantity $\kappa = \frac{M}{m}$ can be seen as the condition number of the cost function, so we obtain essentially the same linear dependence on κ as we did in the convex quadratic case.

The strongly convex optimization setting has a clear advantage relative to our analysis of quadratics and linear regression: it applies to cost functions which aren't quadratic. Furthermore, many cost functions we might actually be interested in optimizing satisfy the assumptions, in which case we have formal guarantees on the optimizer we're actually using.

On the other hand, analyzing linear regression as a model system has some distinct advantages:

1. Unregularized linear regression is not strongly convex, so the strongly convex analysis is not strictly a generalization of our linear regression analysis.

2. One particular feature of strongly convex problems is that they have a unique global optimum which the optimizer coverges to. Hence, the strongly convex analysis is incapable of explaining the implicit regularization and double descent effects, which require understanding the detailed dynamics to determine which stationary point is converged to.

3. The linear regression analysis makes more specific predictions than Theorem 1. E.g., we saw that faster convergence along a particular direction can be either a feature or a bug (see Sction 3.1). The linear regression analysis lets us understand *which* directions train faster, an insight that can't be obtained from the strongly convex analysis. (A reader who already has a sophisticated intuition for optimization might be able to deduce similar conclusions by reading the proof, but this probably depends on already having worked through examples like the quadratic case.)

4. We could carry out all the steps of the linear regression analysis without guessing the answer in advance. Each step we went through is entirely natural, or should seem so by the end of this course. By contrast, the strongly convex optimization proof is something you'd come up with post-hoc after having thought hard about the problem. In fact, an optimization researcher would probably come up with such a proof by way of reasoning through model problems including quadratics. It's just that all of this reasoning is hidden once the abstract result is formulated.

As you've probably guessed, I tend to favor model systems. It's very hard to formulate rigorous proofs in abstract settings when the assumptions are general enough to capture real neural networks. And once this is done, the proof is often so complicated that it obscures the underlying ideas. On the other hand, for any neural net phenomenon, there's probably a model system that we can analyze in detail in order to get better understanding.

Of course, mathematical analyses of simple toy problems don't directly prove anything about the neural nets we train in practice. (And for that matter, neither do proofs involving idealized neural net training scenarios!) But these idealized analyses are very useful for making *educated guesses* about what might happen for neural net training, and many of the important advances in neural nets were invented precisely by reasoning through toy examples.

If we truly want to understand what is happening for real neural nets, we need to go a step further, and validate the predictions made by the model system. Just like with any other experimental science, the way to do this is to make lots of specific quantitative and qualitative predictions — e.g. predict how one measurement changes as a result of changing some independent variable — and then compare these predictions against the behavior of real networks. Since simpler model systems often yield more predictions than complicated ones, simplistic toy problems can often be *more* useful for understanding neural net behavior than more abstract settings with supposedly weaker or more realistic assumptions.

6 Discussion

This is supposed to be a course on *neural net* training dynamics. Why did we just spend an entire lecture on linear regression? There are multiple reasons:

1. Linear regression is an important model system, and can give us a lot of intuition for how neural nets behave. Analyzing linear regression doesn't directly prove anything about neural nets, but it lets us formulate hypotheses to explain phenomena we've observed. Sometimes it yields testable predictions beyond the original phenomenon, and we can use this to validate the explanation for neural nets.

   In this sense, linear regression is part of a toolbox of model problems which also includes, e.g., noisy quadratic objectives, linear neural networks, Gaussian processes, matrix completion, and bilinear games.

2. We can approximate a nonlinear system by taking a Taylor approximation around some point of interest (e.g. the initialization or the optimal solution). A second-order Taylor approximation of a cost function around the optimum reduces the problem to a convex quadratic, so this lecture's analysis directly applies. In classical mechanics, taking a second-order Taylor approximation to a potential energy results in a harmonic oscillator; so the harmonic oscillator is universal. This lecture's gradient descent dynamics are universal in the same sense.

3. Amazingly, in certain situations deep neural networks have been shown to behave like linear regression on a high-dimensional random feature space. This surprising property allows us to prove optimization and generalization bounds for real neural nets. Admittedly, the settings where this theory applies don't *quite* match the state-of-the-art on real applications, but it comes close enough that it's interesting to try to understand exactly how the theory breaks down. This is currently a major topic of interest in deep learning theory.

References

Milkhail 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*, 2019.

Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. arXiv:1903.08560v4, 2019.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

Yann LeCun. Efficient backprop. In *Neural Networks: Tricks of the Trade*. Springer, 1998.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. arXiv:1912.02292v1, 2019.