10

Kernel Learning

I protest against the use of an infinite quantity as an actual entity; this is never allowed in mathematics. The infinite is only a manner of speaking . . . .

Carl Friedrich Gauss [61].

Now that we know essentially everything we can possibly know about the initialization distribution of the preactivations *and the NTK*, it's finally time to learn *with gradients*!

In this chapter, we'll analyze the training of infinite-width neural networks by gradient-descent optimization. Of course, the infinite-width network is really only a manner of speaking, and we cannot actually instantiate one in practice. However, as we saw from our previous finite-width analyses, they can still provide a useful *model* of an actual entity when the depth-to-width ratio is sufficiently small.

Thus, the analysis of such networks is important for two reasons. First, this limit can tell us a lot about the correct scaling and tuning of our various hyperparameters; we've already seen this previously as our criticality analysis always begins at infinite width. Second, since our finite-width analysis is perturbative in $1/n$, understanding the infinite-width limit is a prerequisite for us to understand learning with more realistic finite-width networks in §11 and §∞. With those remarks in mind, let's preview our analysis of gradient-based learning at infinite width.

Just as a new biological neural network begins its journey by taking a small step, so does a freshly-initialized artificial neural network. In §10.1 we'll take such a step, observing that the gradient-descent training of infinite-width networks is simply described by the frozen NTK and that the change in the network outputs can be consistently truncated to linear order in the global learning rate. This simplicity leads us first to observe that the network's output components move independently of each other (§10.1.1) and then second to find an absence of representation learning in the hidden layers (§10.1.2). At this point you might have an uncanny sense of déjà vu, as we found the exact same limitations in §6.3 for infinite-width networks that learn via exact Bayesian inference.

After a small step comes a giant leap. In §10.2 we'll make a large parameter update and find a closed-form solution for a *fully-trained* infinite-width network. For these

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

networks, such a solution memorizes the entire training set, and we'll show that this solution is the same regardless of whether we reach it in one Newton step (§10.2.1) or many steps of (stochastic) gradient descent (§10.2.2), and doesn't depend on the form of the loss that we use (§10.2.3).

In fact, in §10.2.4 we'll see that the prediction of a *particular* fully-trained infinite-width network on unseen test inputs is fixed entirely by the initial network output, the frozen NTK, and the contents of the training set. To analyze this, we evaluate the statistics of the associated *ensemble*, identifying the mean (neural tangent) kernel prediction as well as the covariance of that prediction across different realizations. Recalling our discussion of approximate methods for Bayesian model fitting in §6.2.1, we are now able to make more precise the connection between gradient-based learning and maximum likelihood estimation by discussing the sense in which our distribution over fully-trained infinite-width networks is a generalized posterior distribution.

In §10.3, we'll put the predictions of these fully-trained infinite-width networks to the test. Here we'll introduce the quantitative measure of training success, the *generalization error*, and decompose it into a *bias* term and a *variance* term. The former term compares the mean predictions of the ensemble on the test inputs to the true function values from the test set, while the latter term measures the instantiation-to-instantiation fluctuations of that prediction across different fully-trained networks in our ensemble.

Naturally, there is a tradeoff between these bias and variance terms, corresponding to our preference for the ensemble to contain networks that are both flexible *and* confident. By explicitly working out the generalization error when a test input is near one of the training samples in §10.3.1, we'll see how balancing such a tradeoff gives a prescription for tuning the initialization hyperparmeters, via the principle of criticality, and for tuning the training hyperparameters, according to the learning rate equivalence principle.

In §10.3.2 we'll extend our analysis to situations where a test input is near two training samples. This will let us understand how our fully-trained networks interpolate and extrapolate, letting us comment on the activation-function-induced inductive bias of the network output in general. In particular, we'll be able to see how nonlinear activation functions are able to nonlinearly interpolate or extrapolate around two training examples.

Finally, in §10.4 we'll take a small step back to give our discussion of infinite-width networks a broader context. In particular, we'll introduce the linear model, one of the simplest models in traditional machine learning, and explain its relationship to another traditional set of algorithms known as kernel methods. This will let us see that infinite-width MLPs are essentially just linear models based on random features and dually let us identify both the infinite-width Bayesian *kernel* and the frozen neural tangent *kernel* with this more traditional notion of a *kernel*.

After discussing the limitations of such kernel methods, you will thoroughly understand the need to go beyond the infinite-width limit so that our effective theory can fully incorporate some of the more exciting properties of practical deep learning models.

10.1 A Small Step

Now let's take our first step in a journey toward the minimum of the loss. We'll begin by considering how the preactivations change in the first step after initialization at $t = 0$.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Recalling that the evolution of any neural-network observable $\mathcal{O}\left(z^{(\ell)}\right)$ is governed by the NTK through the update equation (8.3), we have for an ℓ-th-layer preactivation:

$$\begin{aligned} \text{đ}z_{i;\delta}^{(\ell)} &\equiv z_{i;\delta}^{(\ell)}(t=1) - z_{i;\delta}^{(\ell)}(t=0) \\ &= -\eta \sum_{j=1}^{n_\ell} \sum_{\tilde{\alpha}\in\mathcal{A}} \frac{d\mathcal{L}_{\mathcal{A}}}{dz_{j;\tilde{\alpha}}^{(\ell)}} \widehat{H}_{ij;\tilde{\alpha}\delta}^{(\ell)} + O\left(\eta^2\right). \end{aligned} \tag{10.1}$$

Here, the ℓ-th-layer NTK $\widehat{H}_{ij;\tilde{\alpha}\delta}^{(\ell)}$ and the factor $d\mathcal{L}_{\mathcal{A}}/dz_{j;\tilde{\alpha}}^{(\ell)}$ are both evaluated at initialization; from now on we'll drop the explicit dependence on the number of steps t when a quantity is being evaluated at initialization $t = 0$, unless we want to emphasize it for clarity. Henceforth, we'll use the prefix đ to indicate the *update* to a quantity after the first step of gradient descent.

Further, in writing (10.1) we have resurrected the sample-index notation of alpha-with-tilde for the inputs in the training set $\tilde{\alpha} \in \mathcal{A}$, and we will soon resurrect beta-with-dot for inputs in the test set $\dot{\beta} \in \mathcal{B}$; as before, we'll also use delta-with-no-decoration for generic inputs in either set: $\delta \in \mathcal{D} = \mathcal{A} \cup \mathcal{B}$. Thus, to be explicitly clear, the update (10.1) gives the change after the first gradient descent training step in an ℓ-th-layer preactivation evaluated on a sample from either the test set or the training set.

Now, let's specialize to the infinite-width limit. Recall in this limit that the NTK self-averages, such that the NTK for *any* particular realization of the network parameters will be equal to the infinite-width NTK mean, which we have been calling the *frozen NTK*: $\widehat{H}_{ij;\tilde{\alpha}\delta}^{(\ell)} = \delta_{ij}\Theta_{\tilde{\alpha}\delta}^{(\ell)} + O(1/n)$. With this in mind, the update equation at infinite width simplifies to

$$\text{đ}z_{i;\delta}^{(\ell)} = -\eta \sum_{\tilde{\alpha}\in\mathcal{A}} \Theta_{\delta\tilde{\alpha}}^{(\ell)} \frac{d\mathcal{L}_{\mathcal{A}}}{dz_{i;\tilde{\alpha}}^{(\ell)}} + O\left(\frac{1}{n}\right). \tag{10.2}$$

Here, the update does not mix neural indices as the mean of the NTK is diagonal in those indices, while the presence of off-diagonal terms in the frozen NTK would indicate that information from one training sample informs the update of another. Note importantly that we have purposefully truncated the $+O(\eta^2)$ part of (10.1), which contains higher-order corrections to the update from the series expansion in the global learning rate η; in §11 we'll explicitly analyze these $O(\eta^2)$ terms and show that they are suppressed by $1/n$. Thus, in the strict infinite-width limit they identically vanish, making the linear truncation exact.

In this section, we'll take a look at what such an infinite-width small-step update entails for the network outputs with $\ell = L$ (§10.1.1) and for preactivations in the final hidden layer with $\ell = L - 1$ (§10.1.2).[1] These analyses will more or less parallel §6.3.2 and §6.3.3, where we considered the posterior distribution for infinite-width networks updated via exact Bayesian inference.

[1] After reading the next section, it should be clear that the results here are true even at the minimum of the loss at the end of training. We say this now to head off any potential objections of the form, "What if there is some number of steps t for which the quantity $\eta t/n$ is of order one?"

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

10.1.1 No Wiring

Specializing to the network output $z_{i;\delta}^{(L)}$, the update (10.2) simply becomes

$$đz_{i;\delta}^{(L)} = -\eta \sum_{\tilde{\alpha}\in\mathcal{A}} \Theta_{\delta\tilde{\alpha}}^{(L)} \epsilon_{i;\tilde{\alpha}}, \tag{10.3}$$

where we recall the now-familiar error factor defined in (7.16) as

$$\epsilon_{i;\tilde{\alpha}} \equiv \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_{i;\tilde{\alpha}}^{(L)}}. \tag{10.4}$$

We're going to extensively analyze this update to network outputs in §10.2 and onwards. Here, let us just point out that the update to the i-th feature $z_{i;\delta}^{(L)}(t=1)$ depends only on the i-th component of the error factor $\epsilon_{i;\tilde{\alpha}}$. This mirrors the phenomenon of *no wiring* for the network outputs that we observed in §6.3.2 for the exact Bayesian inference at infinite width.

To be more concrete, for the MSE loss (7.2),

$$\mathcal{L}_{\mathcal{A}} \equiv \frac{1}{2} \sum_{i=1}^{n_L} \sum_{\tilde{\alpha}\in\mathcal{A}} \left(z_{i;\tilde{\alpha}}^{(L)} - y_{i;\tilde{\alpha}} \right)^2, \tag{10.5}$$

the error factor is simply given by the difference between the true output and the initial output, $\epsilon_{i;\tilde{\alpha}} = z_{i;\tilde{\alpha}}^{(L)} - y_{i;\tilde{\alpha}}$. We thus see that all the output components move independently from each other, and there's no way for correlations between these components to be learned.[2] In addition, since (10.3) is a stochastic equation describing the update to any particular network in the ensemble, there is no wiring for any particular realization of a one-step-into-training infinite-width network.

10.1.2 No Representation Learning

We'll have to work a little harder to analyze the update to the preactivations in the penultimate layer $z_{i;\delta}^{(L-1)}$. To start, we can evaluate the derivative of the loss in the update equation (10.2) using the *backward* equation (8.17):

$$\frac{d\mathcal{L}_{\mathcal{A}}}{dz_{j;\tilde{\alpha}}^{(L-1)}} = \sum_{i=1}^{n_L} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_{i;\tilde{\alpha}}^{(L)}} \frac{dz_{i;\tilde{\alpha}}^{(L)}}{dz_{j;\tilde{\alpha}}^{(L-1)}} = \sum_{i=1}^{n_L} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_{i;\tilde{\alpha}}^{(L)}} W_{ij}^{(L)} \sigma'^{(L-1)}_{j;\tilde{\alpha}}. \tag{10.6}$$

[2] As another example, we can take a cross-entropy loss of the form $\mathcal{L}_{\mathcal{A}} = -\sum_{j,\tilde{\alpha}} p_{j;\tilde{\alpha}} \log(q_{j;\tilde{\alpha}})$; feel free to flip forward and look at (10.36). In this case we have a target distribution $p_{i;\tilde{\alpha}}$ for which we want to fit the *softmax* distribution – cf. (6.14) – of the network outputs $q_{i;\tilde{\alpha}} \equiv \exp(z_{i;\tilde{\alpha}}^{(L)}) / \left[\sum_k \exp(z_{k;\tilde{\alpha}}^{(L)})\right]$. After noting that $\partial q_{j;\tilde{\alpha}} / \partial z_{i;\tilde{\alpha}}^{(L)} = (\delta_{ij} - q_{i;\tilde{\alpha}}) q_{j;\tilde{\alpha}}$, we find for the error factor

$$\epsilon_{i;\tilde{\alpha}} = \sum_j \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial q_{j;\tilde{\alpha}}} \frac{\partial q_{j;\tilde{\alpha}}}{\partial z_{i;\tilde{\alpha}}^{(L)}} = -\sum_j \frac{p_{j;\tilde{\alpha}}}{q_{j;\tilde{\alpha}}} q_{j;\tilde{\alpha}} (\delta_{ij} - q_{i;\tilde{\alpha}}) = -p_{i;\tilde{\alpha}} + \left(\sum_j p_{j;\tilde{\alpha}} \right) q_{i;\tilde{\alpha}} = q_{i;\tilde{\alpha}} - p_{i;\tilde{\alpha}}. \tag{10.7}$$

Therefore, in this case too we see that the error factor $\epsilon_{i;\tilde{\alpha}}$ depends only on the i-th component of the softmax distribution, and no correlation between output components will be generated.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Substituting this into the update (10.2) at $\ell = L - 1$, we get a stochastic equation describing the change in the final hidden-layer representation for any particular network:

$$đz_{j;\delta}^{(L-1)} = -\eta \sum_{i=1}^{n_L} \sum_{\tilde{\alpha}\in\mathcal{A}} \Theta_{\delta\tilde{\alpha}}^{(L-1)} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_{i;\tilde{\alpha}}^{(L)}} W_{ij}^{(L)} \sigma'^{(L-1)}_{j;\tilde{\alpha}}. \tag{10.8}$$

To make progress here, we're going to have to analyze the distribution over such updates.

First, the mean update is given by

$$\mathbb{E}\left[đz_{j;\delta}^{(L-1)}\right] = -\eta \sum_{i=1}^{n_L} \sum_{\tilde{\alpha}\in\mathcal{A}} \Theta_{\delta\tilde{\alpha}}^{(L-1)} \mathbb{E}\left[\frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z_{i;\tilde{\alpha}}^{(L)}} W_{ij}^{(L)} \sigma'^{(L-1)}_{j;\tilde{\alpha}}\right]. \tag{10.9}$$

This expectation involves an interlayer correlation between the error factor $\partial \mathcal{L}_{\mathcal{A}}/\partial z_{i;\tilde{\alpha}}^{(L)}$ from the L-th layer and the derivative of the activation $\sigma'^{(L-1)}_{j;\tilde{\alpha}}$ from the $(L-1)$-th layer, in addition to a weight insertion $W_{ij}^{(L)}$. From our previous experience we know that such interlayer expectations are suppressed by a factor of $1/n$, vanishing in the strict infinite-width limit. To be extra careful, let's compute the mean explicitly.

To do so, recall our generating function for interlayer correlations (8.53) and specialize to the penultimate layer $\ell = L - 1$. (You'll probably want to flip back and remind yourself.) Since we have not previously evaluated the case with a single weight insertion, let's calculate and record it here. Differentiating the generating function once with respect to the source as $\frac{d}{d\mathcal{J}_{ij}}$ and then setting the source to zero, we get

$$\mathbb{E}\left[\mathcal{O}\left(z^{(L)}\right) W_{ij}^{(L)} \mathcal{Q}\left(z^{(L-1)}\right)\right] = \frac{C_W^{(L)}}{n_{L-1}} \sum_{\delta\in\mathcal{D}} \mathbb{E}\left[\left\langle\!\!\left\langle \frac{\partial \mathcal{O}}{\partial z_{i;\delta}^{(L)}} \right\rangle\!\!\right\rangle_{\widehat{G}^{(L)}} \sigma_{j;\delta}^{(L-1)} \mathcal{Q}\left(z^{(L-1)}\right)\right]. \tag{10.10}$$

Applying this formula to the above expression for our update (10.9), we get

$$\begin{aligned}\mathbb{E}\left[đz_{j;\delta}^{(L-1)}\right] &= -\eta \frac{C_W^{(L)}}{n_{L-1}} \sum_{i=1}^{n_L} \sum_{\tilde{\alpha}_1\tilde{\alpha}_2\in\mathcal{A}} \Theta_{\delta\tilde{\alpha}_1}^{(L-1)} \mathbb{E}\left[\left\langle\!\!\left\langle \frac{\partial^2 \mathcal{L}_{\mathcal{A}}}{\partial z_{i;\tilde{\alpha}_1}^{(L)} \partial z_{i;\tilde{\alpha}_2}^{(L)}} \right\rangle\!\!\right\rangle_{\widehat{G}^{(L)}} \sigma_{j;\tilde{\alpha}_2}^{(L-1)} \sigma'^{(L-1)}_{j;\tilde{\alpha}_1}\right] \qquad (10.11)\\ &= -\eta \frac{C_W^{(L)}}{n_{L-1}} \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} \Theta_{\delta\tilde{\alpha}_1}^{(L-1)} \sum_{i=1}^{n_L} \left\langle\!\!\left\langle \frac{\partial^2 \mathcal{L}_{\mathcal{A}}}{\partial z_{i;\tilde{\alpha}_1}^{(L)} \partial z_{i;\tilde{\alpha}_2}^{(L)}} \right\rangle\!\!\right\rangle_{K^{(L)}} \left\langle \sigma'_{\tilde{\alpha}_1} \sigma_{\tilde{\alpha}_2} \right\rangle_{K^{(L-1)}} + O\left(\frac{1}{n^2}\right)\end{aligned}$$

and see that this expression is manifestly suppressed by $1/n$. Thus, on average across our ensemble, we have found that there's no representation learning in the penultimate layer in the infinite-width limit.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Next, let's consider the variance of the update (10.8):

$$\begin{aligned}
&\mathbb{E}\left[đz^{(L-1)}_{j_1;\delta_1} đz^{(L-1)}_{j_2;\delta_2}\right] \qquad (10.12)\\
&\quad = \eta^2 \sum_{i_1,i_2=1}^{n_L} \sum_{\tilde{\alpha}_1\tilde{\alpha}_2\in\mathcal{A}} \Theta^{(L-1)}_{\delta_1\tilde{\alpha}_1}\Theta^{(L-1)}_{\delta_2\tilde{\alpha}_2}\, \mathbb{E}\left[\frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z^{(L)}_{i_1;\tilde{\alpha}_1}} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z^{(L)}_{i_2;\tilde{\alpha}_2}} W^{(L)}_{i_1 j_1} W^{(L)}_{i_2 j_2} \sigma'^{(L-1)}_{j_1;\tilde{\alpha}_1} \sigma'^{(L-1)}_{j_2;\tilde{\alpha}_2}\right]\\
&\quad = \delta_{j_1 j_2}\, \eta^2 \frac{C^{(L)}_W}{n_{L-1}} \sum_{\tilde{\alpha}_1\tilde{\alpha}_2\in\mathcal{A}} \Theta^{(L-1)}_{\delta_1\tilde{\alpha}_1}\Theta^{(L-1)}_{\delta_2\tilde{\alpha}_2} \sum_{i=1}^{n_L} \left\langle\!\!\left\langle \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z^{(L)}_{i_1;\tilde{\alpha}_1}} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z^{(L)}_{i_2;\tilde{\alpha}_2}} \right\rangle\!\!\right\rangle_{K^{(L)}} \left\langle \sigma'_{\tilde{\alpha}_1}\sigma'_{\tilde{\alpha}_2}\right\rangle_{K^{(L-1)}}\\
&\qquad + O\left(\frac{1}{n^2}\right) .
\end{aligned}$$

In the last step, we applied our interlayer correlation formula with two weight insertions (8.55) and then picked up the leading contribution.[3] With this, we see that the covariance of the update,

$$\mathrm{Cov}\left[đz^{(L-1)}_{j_1;\delta_1},\, đz^{(L-1)}_{j_2;\delta_2}\right] \equiv \mathbb{E}\left[đz^{(L-1)}_{j_1;\delta_1} đz^{(L-1)}_{j_2;\delta_2}\right] - \mathbb{E}\left[đz^{(L-1)}_{j_1;\delta_1}\right]\mathbb{E}\left[đz^{(L-1)}_{j_2;\delta_2}\right] = O\left(\frac{1}{n}\right), \qquad (10.13)$$

is manifestly suppressed by $1/n$, vanishing in the strict infinite-width limit. Since the distribution of updates to the penultimate-layer preactivations has vanishing mean and covariance, we conclude that the distributions before and after the learning update are equal. Mirroring what we found for exact Bayesian inference in §6.3.3, there's no representation learning for gradient-based learning in the infinite-width limit.[4]

10.2 A Giant Leap

That's one small step for [a] machine, one giant leap for AI.

Neil AI-Strong

In the last section, we started to understand training for infinite-width networks by taking a small step of gradient descent. Of course, what we'd actually like is to understand the behavior of fully-trained networks at the minimum of their losses. Naturally, we could continue by taking many many small steps until our networks are fully trained, and indeed this is how networks are typically trained in practice.

That said, in §10.2.1 we'll first show that we can actually fully train infinite-width networks in *one* theoretical gradient-descent step. That is, we can take a *giant leap* right to the minimum of the loss. We'll then explain in §10.2.2 that the theoretical minimum

[3] Note that the second term in (8.55) is of order $1/n^2$ and hence subleading.

[4] You can check the higher-order connected correlators of the update distribution, if you'd like. However, as we already said before we started these computations, these sorts of interlayer correlations are naturally suppressed by factors of $1/n$, and so will be the higher-order connected correlators.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

we've found by our giant leap is the same minimum we would have found in practice by taking many steps of gradient descent, or even by using *stochastic gradient descent* with decreasing learning rates. This equivalence makes our giant leap a powerful theoretical tool. After a brief aside about the cross-entropy loss in §10.2.3, finally in §10.2.4 we'll see how our fully-trained infinite-width networks make predictions on previously unseen examples, though a detailed analysis of these test-set predictions will be postponed until the following section.

10.2.1 Newton's Method

Our first goal is to find a single step such that the network outputs equal the true outputs,

$$z^{(L)}_{i;\tilde{\alpha}}(t=1) = y_{i;\tilde{\alpha}}, \tag{10.14}$$

for all samples $x_{\tilde{\alpha}}$ in the training set $\tilde{\alpha} \in \mathcal{A}$. This condition will be our definition of *fully trained*, and it's easy to see that such a condition will minimize the training loss for any of the loss functions that we've described. Recalling the gradient-descent update for neural-network outputs (10.3) and rearranging terms, we see that our giant-leap update must satisfy

$$z^{(L)}_{i;\tilde{\alpha}_1} - y_{i;\tilde{\alpha}_1} = \eta \sum_{\tilde{\alpha}_2 \in \mathcal{A}} \widetilde{\Theta}^{(L)}_{\tilde{\alpha}_1\tilde{\alpha}_2} \frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z^{(L)}_{i;\tilde{\alpha}_2}} \tag{10.15}$$

for the network to be fully trained.

As a reminder, our convention is that quantities without an explicit step argument are evaluated at the point of initialization $t = 0$; in particular, the constraint (10.15) is written solely in terms of the quantities at initialization. Additionally, note that the tilde on $\widetilde{\Theta}^{(L)}_{\tilde{\alpha}_1\tilde{\alpha}_2}$ emphasizes that it's an $N_{\mathcal{A}} \times N_{\mathcal{A}}$-dimensional submatrix of the full frozen NTK matrix $\Theta^{(L)}_{\delta_1\delta_2}$ evaluated on pairs of inputs $(x_{\tilde{\alpha}_1}, x_{\tilde{\alpha}_2})$ in the training set $\mathcal{A}$ *only*. This emphasis will soon prove itself useful, as it did before in §6.

How can we satisfy our giant-leap condition (10.15)? Since the left-hand side is exactly the error factor of the *MSE loss* (10.5), let's first specialize to the MSE loss. Plugging in (10.5) for $\mathcal{L}_{\mathcal{A}}$, we get a concrete equation to solve:

$$z^{(L)}_{i;\tilde{\alpha}_1} - y_{i;\tilde{\alpha}_1} = \sum_{\tilde{\alpha}_2 \in \mathcal{A}} \eta \widetilde{\Theta}^{(L)}_{\tilde{\alpha}_1\tilde{\alpha}_2} \left(z^{(L)}_{i;\tilde{\alpha}_2} - y_{i;\tilde{\alpha}_2} \right). \tag{10.16}$$

However, for generic neural networks, the frozen NTK $\Theta^{(L)}_{\tilde{\alpha}_1\tilde{\alpha}_2}$ will have nonzero off-diagonal components mixing different sample indices. This unfortunately means that the condition (10.16) is impossible to satisfy by tuning the single global learning rate η.

Said another way, the issue is that our global learning rate η is a *scalar*, but here we need it to be *tensor* in order to undo the mixing of the sample indices by the frozen NTK. To enable this, we need to further generalize gradient descent. In our first extension of

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

the gradient descent algorithm (7.11) – discussed under the heading *Tensorial Gradient Descent* – we introduced a *learning-rate tensor* on parameter space,

$$\eta \to \eta\lambda_{\mu\nu}, \tag{10.17}$$

which let us mediate how each model parameter individually contributes to the gradient-descent update of the others and let us take steps with unequal magnitudes in various directions in parameter space. The consequence of having such a learning-rate tensor was integrated into the definition of the NTK and then informed our analyses in §7–§9.[5]

Now, we need to further extend this generalization to sample indices as

$$\eta\lambda_{\mu\nu} \to \eta\lambda_{\mu\nu}\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}, \tag{10.18}$$

where we have introduced a new symmetric matrix $\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}$ that we will call the **Newton tensor**. This enables us to take an anisotropic step in sample space as well. Specifically, we extend the parameter update equation (7.11) to

$$\text{đ}\theta_\mu \equiv \theta_\mu(t=1) - \theta_\mu(t=0) = -\sum_{\nu,\tilde{\alpha}_1,\tilde{\alpha}_2,i} \eta\lambda_{\mu\nu}\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2} \frac{dz^{(L)}_{i;\tilde{\alpha}_1}}{d\theta_\nu} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z^{(L)}_{i;\tilde{\alpha}_2}}, \tag{10.19}$$

which we will call a **second-order update**.[6] Plugging this second-order update into our expansion for the network outputs, we get

$$\begin{aligned} z^{(L)}_{i;\delta_1}(t=1) &= z^{(L)}_{i;\delta_1} + \sum_\mu \frac{dz^{(L)}_{i;\delta_1}}{d\theta_\mu}\text{đ}\theta_\mu + O\left(\frac{1}{n}\right) \\ &= z^{(L)}_{i;\delta_1} - \eta \sum_{\tilde{\alpha}_2,\tilde{\alpha}_3\in\mathcal{A}} \Theta^{(L)}_{\delta_1\tilde{\alpha}_2}\kappa^{\tilde{\alpha}_2\tilde{\alpha}_3} \frac{\partial \mathcal{L}_\mathcal{A}}{\partial z^{(L)}_{i;\tilde{\alpha}_3}} + O\left(\frac{1}{n}\right). \end{aligned} \tag{10.20}$$

Substituting this update into our fully-trained condition (10.14) and still using the MSE loss, we get a new constraint

$$z^{(L)}_{i;\tilde{\alpha}_1} - y_{i;\tilde{\alpha}_1} = \sum_{\tilde{\alpha}_2,\tilde{\alpha}_3\in\mathcal{A}} \left(\eta\widetilde{\Theta}^{(L)}_{\tilde{\alpha}_1\tilde{\alpha}_2}\kappa^{\tilde{\alpha}_2\tilde{\alpha}_3}\right)\left(z^{(L)}_{i;\tilde{\alpha}_3} - y_{i;\tilde{\alpha}_3}\right). \tag{10.21}$$

We'll satisfy this constraint shortly.

[5] Most importantly, this let us scale the effective learning rate differently for the biases and weights; we saw in §8.0 and then in §9.4 that this was essential for ensuring that both parameter groups get properly trained. Also, please remember that, even when written generally as $\lambda_{\mu\nu}$, our learning-rate tensor is restricted such that it does not mix parameters from different layers.

[6] The name of this update descends from the fact that similar updates are used to define optimization algorithms that incorporate information from the second derivative of the loss. Such algorithms are generally called *second-order methods*. We will show shortly that this new algorithm minimizes the loss just as well (better, actually).

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

For a different perspective on this new second-order update, rather than modifying the optimization algorithm, we can instead find the same constraint (10.21) by adopting a different loss. Consider a *generalized* MSE loss

$$\mathcal{L}_{\mathcal{A},\kappa}(\theta) = \frac{1}{2}\sum_{i=1}^{n_L}\sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} \kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}\left(z^{(L)}_{i;\tilde{\alpha}_1} - y_{i;\tilde{\alpha}_1}\right)\left(z^{(L)}_{i;\tilde{\alpha}_2} - y_{i;\tilde{\alpha}_2}\right), \tag{10.22}$$

where here the Newton tensor $\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}$ acts as a *metric* on sample space.[7] For this loss, the derivative with respect to the network output – i.e., the error factor – is now given by

$$\frac{\partial \mathcal{L}_{\mathcal{A}}}{\partial z^{(L)}_{i;\tilde{\alpha}_2}} = \sum_{\tilde{\alpha}_3\in\mathcal{A}} \kappa^{\tilde{\alpha}_2\tilde{\alpha}_3}\left(z^{(L)}_{i;\tilde{\alpha}_3} - y_{i;\tilde{\alpha}_3}\right). \tag{10.23}$$

Substituting this error factor into our condition for being fully trained, (10.15), we find the same constraint (10.21) using a standard gradient-descent update with our generalized MSE loss (10.22) as we did just before using our second-order update (10.19) with the standard MSE loss. Either perspective is a valid way to think about our theoretical optimization and, as we will explain more generally in §10.2.2, any of these choices of algorithms and losses will lead to the same fully-trained network.

Now, let's find the solution to our giant-leap constraint (10.21). By inspection, this is satisfiable if we can set the term in the first parenthesis to the identity matrix,

$$\sum_{\tilde{\alpha}_2\in\mathcal{A}} \eta\widetilde{\Theta}^{(L)}_{\tilde{\alpha}_1\tilde{\alpha}_2}\kappa^{\tilde{\alpha}_2\tilde{\alpha}_3} = \delta_{\tilde{\alpha}_1}^{\ \tilde{\alpha}_3}, \tag{10.24}$$

which we can ensure by setting the product of the global learning rate and the Newton tensor as

$$\eta\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2} = \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)}. \tag{10.25}$$

Here, the object on the right-hand side is the *inverse* of the $N_{\mathcal{A}} \times N_{\mathcal{A}}$-dimensional submatrix $\widetilde{\Theta}^{(L)}_{\tilde{\alpha}_1\tilde{\alpha}_2}$, which as a reminder is evaluated on pairs of inputs $(x_{\tilde{\alpha}_1}, x_{\tilde{\alpha}_2})$ in the training set $\mathcal{A}$ *only* and is defined via the equation

$$\sum_{\tilde{\alpha}_2\in\mathcal{A}} \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)}\,\widetilde{\Theta}^{(L)}_{\tilde{\alpha}_2\tilde{\alpha}_3} = \delta^{\tilde{\alpha}_1}_{\ \tilde{\alpha}_3}. \tag{10.26}$$

Similarly to our work in §6.3 on infinite-width Bayesian inference, the decoration of these submatrices with tildes is useful in order to clearly distinguish these submatrices from

[7] Similarly, we could have taken the perspective that the learning-rate tensor $\lambda_{\mu\nu}$ acts as a metric on *parameter space*.

Note also that with this interpretation, the standard MSE loss is just the generalized MSE loss with the Euclidean metric $\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2} \to \delta^{\tilde{\alpha}_1\tilde{\alpha}_2}$. In some sense, it's more pleasing to write it this way if you're familiar with general relativity; writing the Newton tensor with sample indices *raised* allows us to adopt a rule of only summing over sample indices when they come in a raised-lowered pair. Similarly, note that the insertion of the Newton tensor in our second-order update (10.19) follows this pattern as well.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

submatrices that also involve the *test set* $\mathcal{B}$. Also, as before for the kernel and its inverse, we will always denote the NTK inverse by an object with sample indices *raised*.

The algorithm with the particular choice (10.25) is known as **Newton's method** (which acausally explains why we called $\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}$ the *Newton tensor*).[8] With it, we can simply write down a solution that fully trains the network in one step:

$$\theta^\star_\mu = \theta_\mu(t=0) - \sum_{\nu,\tilde{\alpha}_1,\tilde{\alpha}_2,i} \lambda_{\mu\nu} \frac{dz^{(L)}_{i;\tilde{\alpha}_1}}{d\theta_\nu} \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)} \left(z^{(L)}_{i;\tilde{\alpha}_2} - y_{i;\tilde{\alpha}_2}\right). \tag{10.27}$$

In particular, this is exactly what we'd find by setting the gradient of the loss to zero and solving for the optimal parameters as in (7.7). As we explained back there, such a direct and explicit solution to an optimization problem is only available in special cases, and it turns out that this is precisely the case at infinite width.[9]

Plugging the Newton's method update into our expansion for the network outputs (10.20), we then find the fully-trained network output for a general input $\delta \in \mathcal{D}$:

$$z^{(L)}_{i;\delta}(t=1) = z^{(L)}_{i;\delta} - \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} \Theta^{(L)}_{\delta\tilde{\alpha}_1} \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)} \left(z^{(L)}_{i;\tilde{\alpha}_2} - y_{i;\tilde{\alpha}_2}\right). \tag{10.28}$$

[8] *Newton's method* is a numerical method for finding a zero of a function. For simplicity of presentation, let's take a single-variable function $g(x)$ and suppose that we want to find a solution to the equation $g(x_\star) = 0$; note that this is equivalent to extremizing a function $L(x)$ whose derivative is $g(x)$, i.e., $L'(x) = g(x)$. Newton's method instructs us to start with some guess x_0 and then iterate as

$$x_{t+1} = x_t - \frac{g(x_t)}{g'(x_t)} = x_t - \frac{L'(x_t)}{L''(x_t)}. \tag{10.29}$$

This algorithm is based on making a linear approximation $g'(x_t) \approx [g(x_{t+1}) - g(x_t)]/(x_{t+1} - x_t)$ and then solving for $g(x_{t+1}) = 0$. In general, one needs to iterate (10.29) for several steps in order to get a good approximate solution $x_\star$. When the function is linear as $g(x) = a(x - x_\star)$, however, we get

$$x_1 = x_0 - \frac{a(x_0 - x_\star)}{a} = x_\star, \tag{10.30}$$

for any starting point x_0. Hence Newton's method can land right on the solution in one step, just like our giant leap (10.27) did.

The right-hand side of (10.29) offers another perspective: Newton's method is gradient descent with a "loss" $L(x)$ and a learning rate set as $\eta_t = 1/L''(x_t)$. To see why this is a good choice for the learning rate, let's choose a generic learning rate $x_{t+1} = x_t - \eta_t L'(x_t)$ and Taylor-expand the updated loss $L(x_{t+1})$ to the second order in η_t:

$$L(x_{t+1}) = L(x_t) - \eta_t L'(x_t)^2 + \frac{\eta_t^2}{2} L'(x_t)^2\, L''(x_t) + O\left(\eta_t^3\right). \tag{10.31}$$

Optimizing the learning rate, we see that the truncated expression on the right-hand side is minimized when $\eta_t = 1/L''(x_t)$. In particular, for a quadratic function $L(x) = a(x - x_\star)^2/2$, this truncation is exact, and Newton's method again reaches the minimum in one step. This also makes it clear why optimization algorithms based on Newton's method fall in the class of *second-order methods*: each iteration uses second-order information from the function – the second derivative $L''(x)$ – to set the locally optimal learning rate. Our giant leap expressed in (10.27) and (10.28) is doing exactly that – successfully – for the parameter optimization and for the function approximation, respectively.

[9] We'll later show in §∞.2 that perturbative solutions are possible at finite width.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

In particular, for samples in the training set $\mathcal{A}$, the network output equals the true output $z^{(L)}_{i;\tilde{\alpha}}(t=1) = y_{i;\tilde{\alpha}}$, satisfying our condition for the network to be fully trained (10.14). In other words, our network has perfectly memorized the entire training set.[10] As such, this solution (10.27) also minimizes *any* loss $\mathcal{L}_{\mathcal{A}}(\theta)$ that is minimized by setting the network outputs to the true outputs $z^{(L)}(x_{\tilde{\alpha}};\theta) = y_{i;\tilde{\alpha}}$:

$$\theta^{\star}_{\text{Newton}} = \arg\min_{\theta} \mathcal{L}_{\mathcal{A}}(\theta). \tag{10.32}$$

This means that regardless of whether we used the standard MSE loss (10.5) or the generalized MSE loss (10.22) (or an entirely different loss as long as it has a minimum at $z^{(L)}(x_{\tilde{\alpha}};\theta) = y_{i;\tilde{\alpha}}$), our solution (10.28) will faithfully describe the minimum.[11]

10.2.2 Algorithm Independence

Now let's discuss a related – and by now well-anticipated – property of the infinite-width limit: given a particular initialization $z^{(L)}_{i;\delta}(t=0)$ and the frozen NTK $\Theta^{(L)}_{\delta\tilde{\alpha}_1}$, we'll always get to exactly the same minimum (10.28), whether we get there by one giant leap or we get there by a sequence of many small steps. That is, at infinite width we have **algorithm independence**.

Let's suppose that we have taken $T-1$ steps toward the minimum with a global learning rate $\eta(t)$, a Newton tensor $\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}(t)$, and loss $\mathcal{L}_{\mathcal{A}}(t)$, where these quantities will in general depend on the step t. Different choices of $\eta(t)$, $\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}(t)$, and $\mathcal{L}_{\mathcal{A}}(t)$ will lead to different optimization algorithms; included in this class are Newton's method, gradient descent, and stochastic gradient descent (SGD).[12] Iterating the update (10.20), the network outputs accumulate the changes as

$$z^{(L)}_{i;\delta}(T-1) = z^{(L)}_{i;\delta}(t=0) - \sum_{t=0}^{T-2} \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2} \Theta^{(L)}_{\delta\tilde{\alpha}_1}\, \eta(t)\, \kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}(t)\, \epsilon_{i;\tilde{\alpha}_2}(t), \tag{10.33}$$

where $\epsilon_{i;\tilde{\alpha}_2}(t) \equiv \partial\mathcal{L}_{\mathcal{A}}(t)/\partial z^{(L)}_{i;\tilde{\alpha}_2}$ is the error factor for the training loss $\mathcal{L}_{\mathcal{A}}(t)$ evaluated with respect to the network output $z^{(L)}_{i;\tilde{\alpha}}(t)$ at step t.

Let's then suppose that in the next step $t=T$ we reach the true minimum. We can ensure this by taking a Newton step from $z^{(L)}_{i;\delta}(T-1)$ such that

$$z^{(L)}_{i;\delta}(T) = z^{(L)}_{i;\delta}(T-1) - \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2} \Theta^{(L)}_{\delta\tilde{\alpha}_1} \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)} \left[z^{(L)}_{i;\tilde{\alpha}_2}(T-1) - y_{i;\tilde{\alpha}_2} \right], \tag{10.34}$$

[10] Since infinite-width networks have infinite parameters, it shouldn't be surprising that in this limit the network can memorize the finite training set.

[11] Note that the solution (10.27) depends on the network output at initialization $z^{(L)}_{i;\delta}$, which ultimately depend on the initialization of the parameters $\theta_{\text{init}} = \theta(t=0)$. For different initializations, we will reach different solutions (10.27), each of which will minimize the loss given that particular initialization θ_{init}. We'll have more to say about this in §10.2.4.

[12] To see how this includes SGD (7.10), note that we can either restrict the loss to be a summation over a different *batch* $\mathcal{S}_t \subset \mathcal{A}$ at each step t as $\mathcal{L}_{\mathcal{A}}(t) = \mathcal{L}_{\mathcal{S}_t}$, or equivalently we can choose the Newton tensor $\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}(t)$ to project onto the subset $\mathcal{S}_t$.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

where here we set $\eta(T-1)\,\kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}(T-1) = \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)}$ and chose the standard MSE loss at $t = T-1$ with $\epsilon_{i;\tilde{\alpha}_2}(T-1) = z^{(L)}_{i;\tilde{\alpha}_2}(T-1) - y_{i;\tilde{\alpha}_2}$.[13] Plugging in our expression for the network outputs after the first $T-1$ steps, (10.33), we see

$$
\begin{aligned}
z^{(L)}_{i;\delta}(T) &= z^{(L)}_{i;\delta}(t=0) - \sum_{t=0}^{T-2} \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2} \Theta^{(L)}_{\delta\tilde{\alpha}_1}\, \eta(t)\, \kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}(t)\, \epsilon_{i;\tilde{\alpha}_2}(t) \\
&\quad - \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2} \Theta^{(L)}_{\delta\tilde{\alpha}_1} \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)} \left\{ \left[z^{(L)}_{i;\tilde{\alpha}_2}(t=0) - \sum_{t=0}^{T-2} \sum_{\tilde{\alpha}_3,\tilde{\alpha}_4} \widetilde{\Theta}^{(L)}_{\tilde{\alpha}_2\tilde{\alpha}_3}\, \eta(t)\, \kappa^{\tilde{\alpha}_3\tilde{\alpha}_4}(t)\, \epsilon_{i;\tilde{\alpha}_4}(t) \right] - y_{i;\tilde{\alpha}_2} \right\} \\
&= z^{(L)}_{i;\delta}(t=0) - \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2 \in \mathcal{A}} \Theta^{(L)}_{\delta\tilde{\alpha}_1} \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)} \left[z^{(L)}_{i;\tilde{\alpha}_2}(t=0) - y_{i;\tilde{\alpha}_2} \right] , \qquad (10.35)
\end{aligned}
$$

where to go from the second to the third line we made use of the defining equation for the NTK submatrix inverse (10.26), thus enabling the cancellation. What this result shows is that all the details of the training algorithm in the previous steps $\{\eta(t), \kappa^{\tilde{\alpha}_1\tilde{\alpha}_2}(t), \mathcal{L}_{\mathcal{A}}(t)\}_{t=0,\ldots,T-2}$ were erased: the network output $z^{(L)}_{i;\delta}(T)$ after our final step $t = T$ here in (10.35) is exactly the same as the network output reached after one giant leap (10.28).[14]

Thus, in the infinite-width limit the fully-trained solution is determined by *(i)* the frozen NTK $\Theta^{(L)}_{\delta\tilde{\alpha}}$, with its details depending on the *training hyperparameters* in the learning-rate tensor $\lambda_{\mu\nu}$; *(ii)* the initial newtork outputs $z^{(L)}_{i;\delta}(t=0)$, with its distribution depending on the *initialization hyperparameters*; and *(iii)* the true outputs $y_{i;\tilde{\alpha}}$ for the training set $\mathcal{A}$. It doesn't matter which loss function we used, e.g., MSE or cross-entropy, how many steps we took to get to the minimum, or whether we used gradient descent or SGD.[15] Said another way, *algorithm independence* means that these hyperparameters and training set uniquely specify the statistics of fully-trained networks in the

[13] Note that if we had already reached a minimum at step $T-1$, then this last Newton's step in (10.34) would give no change to the network outputs: $z^{(L)}_{i;\delta}(T) = z^{(L)}_{i;\delta}(T-1)$. Thus, our argument also applies to any algorithm that already reached a minimum with $T-1$ other steps, and we do not have to actually apply the Newton step in practice. We'll address this point again in the next-to-next footnote.

[14] In §∞.2.2, we'll explicitly analyze the dynamics of another optimization algorithm – many many steps of vanilla gradient descent (7.11) – and evaluate its corresponding fully-trained solution. As expected by algorithm independence (10.35), in the infinite-width limit this solution agrees completely with other solutions obtained by different training algorithms.

[15] Some additional comments that didn't make the cut for the main body:

- Newton's method is often impractical to implement directly, since we have to compute the inverse of the frozen NTK submatrix evaluated on the entire training set, $\widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)}$, similar to how we had to invert the kernel for Bayesian inference in §6.3.2. Unlike the case of exact Bayesian inference where we were considering the feasibility of the learning algorithm, here the point is that Newton's method is a theoretical tool that lets us describe a fully-trained extremely-wide network, even if the network was trained very practically by a many-step version of (stochastic) gradient descent.
- Often theorists will take the limit of a very small step size and approximate the optimization dynamics with an ordinary differential equation (ODE). Such an approximation is sometimes misleading, and here we see that it's entirely unnecessary.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

infinite-width limit; Newton's method is just a nice theoretical trick to leap right to the solution. In this way, we can use the giant-leap solution (10.28) to study the outcome of all these different optimization algorithms, which is what we'll do after a brief aside about the cross-entropy loss.

10.2.3 *Aside*: Cross-Entropy Loss

Let's take a brief aside to bring the *cross-entropy loss* out of the footnotes and into the main body. In general, the cross-entropy loss for some dataset $\mathcal{D}$ is defined as

$$\mathcal{L}_{\mathcal{D}} = -\sum_{\delta\in\mathcal{D}}\sum_{i=1}^{n_{\text{out}}} p(i|x_\delta)\log[q(i|x_\delta)]\,, \tag{10.36}$$

where $p(i|x_\delta)$ is a discrete distribution over the components i of the true output

$$p(i|x_\delta) \equiv \frac{\exp[y_{i;\delta}]}{\sum_{j=1}^{n_{\text{out}}}\exp[y_{j;\delta}]}, \tag{10.37}$$

and $q(i|x_\delta)$ is similarly a discrete distribution over the components i of the network's output

$$q(i|x_\delta) \equiv \frac{\exp\left[z^{(L)}_{i;\delta}(t)\right]}{\sum_{j=1}^{n_{\text{out}}}\exp\left[z^{(L)}_{i;\delta}(t)\right]}. \tag{10.38}$$

As we mentioned when discussing the categorical hypothesis in the context of Bayesian model fitting in §6.2.1, the discrete distribution used for (10.37) and (10.38) is sometimes referred to as the *softmax* (6.14). The cross-entropy loss (10.36) is a natural measure of the closeness of discrete distributions such as (10.37) and (10.38).[16]

- For SGD to actually converge to a minimum, you need to decrease the learning rate over the course of training, otherwise the network will fluctuate around, but never actually reach, the minimum. Intuitively, this is because at each step the optimization problem does not include the entire training set.
- A curious reader might wonder what happens if one cannot take a final step according to Newton's method, for instance because it's impractical to invert the frozen NTK submatrix. In fact, if you're already close to the minimum at $t = T - 1$, i.e., if you're essentially at the end of training, then the final step to land exactly on the minimum will be extremely small, and the solution (10.35) is a very good approximation of the network before taking this last theoretical jump.

[16] The proper measure of closeness of distributions is really the *Kullback-Leibler (KL) divergence* (A.12), which we will describe in detail in Appendix A. However, the KL divergence $KL[p\,||\,q]$ and the cross-entropy loss (10.36) only differ by a $z^{(L)}$-independent constant, the entropy $\mathcal{S}[p(i|x_\delta)] = -\sum_{\delta\in\mathcal{D}}\sum_{i=1}^{n_{\text{out}}} p(i|x_\delta)\log[p(i|x_\delta)]$ to be exact, and thus the use of one versus the other is identical under any gradient-based learning algorithm. Note also the lack of exchange symmetry in either loss between p and q. The choice in (10.36) is purposeful and reflects the fact that an untrained model is on a different footing than the true distribution from which observations arise, analogous to the asymmetry between the prior and posterior in Bayesian inference.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

In particular, cross-entropy loss is the appropriate choice for *classification*, when we want to sort the input x into one of n_{out} different classes or categories. Accordingly, the softmax distribution (10.38) transforms the model's output vector with n_{out} real components into a discrete probability distribution. In contrast, the MSE loss is the appropriate choice for *regression*, when the function we want to learn is a vector of real numbers.[17] Importantly, when the initialization hyperparameters are tuned to criticality and the training hyperparameters are selected according to the learning rate equivalence principle, *both losses will be completely well behaved during training.*

When using the cross-entropy loss, typically the true outputs for the training set are given in terms of the softmax values $p(i|x_{\tilde{\alpha}})$ rather than in terms of continuous vectors $y_{i;\tilde{\alpha}}$. Even more typically, the values $p(i|x_{\tilde{\alpha}})$ specify a particular *label*, $i = i^\star_{\tilde{\alpha}}$, with *absolute certainty*, $p(i^\star_{\tilde{\alpha}}|x_{\tilde{\alpha}}) = 1$, while the rest of the components vanish, $p(i|x_{\tilde{\alpha}}) = 0$ for $i \neq i^\star_{\tilde{\alpha}}$; this is known as *hard labeling* or **one-hot encoding** and puts the true value of the network output $y_{i^\star_{\tilde{\alpha}};\tilde{\alpha}}$ at infinity. In such case, no finite amount of training will actually reach the minimum, and in practice as you approach such a minimum, the generalization of the network becomes worse and worse. To remedy this, **early stopping** of the training algorithm is used as a regularization technique to effectively get finite targets $y_{i^\star_{\tilde{\alpha}};\tilde{\alpha}}$.[18]

Now let's specialize to the current context of training neural networks in the infinite-width limit (and assume some kind of regularization is used as described above). In the last section, we noted that any loss that's minimized by setting the network outputs to the true outputs for the training set, $z^{(L)}(x_{\tilde{\alpha}};\theta) = y_{i;\tilde{\alpha}}$, is described at the minimum by the Newton's method giant-leap solution (10.32). It's easy to check that the cross-entropy loss (10.36) is minimized when $q(i|x_\delta) = p(i|x_\delta)$, and a quick inspection of (10.37) and (10.38) shows that this is obtained by the condition $z^{(L)}(x_{\tilde{\alpha}};\theta) = y_{i;\tilde{\alpha}}$.

We do need to make one additional important remark for the cross-entropy loss. Since in this setting we specify the true output in terms of a softmax $p(i|x_{\tilde{\alpha}})$ rather than an n_L-component vector of real numbers $y_{i;\tilde{\alpha}}$, there is an ambiguity in how to set the network output $z^{(L)}_{i;\tilde{\alpha}}$: any component-independent shift $y_{i;\tilde{\alpha}} \to y_{i;\tilde{\alpha}} + c_{\tilde{\alpha}}$ keeps the target distribution $p(i|x_\delta)$ invariant. However, since in this case we care not about the network outputs $z^{(L)}_{i;\delta}$ but rather their softmax $q(i|x_\delta)$ (10.38), this ambiguity doesn't matter in the end. In particular, a shift $y_{i;\tilde{\alpha}} \to y_{i;\tilde{\alpha}} + c_{\tilde{\alpha}}$ in the giant-leap solution (10.35) shifts all the output components by the same amount for each input x_δ, $\sum_{\tilde{\alpha}_1,\tilde{\alpha}_2 \in \mathcal{A}} \Theta^{(L)}_{\delta\tilde{\alpha}_1} \widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)} c_{\tilde{\alpha}_2}$,

[17] We can think of each loss as descending from a different Bayesian hypothesis, cf. our discussion of the uncertain hypothesis and the MSE loss (6.10) and the categorical hypothesis and the softmax distribution (6.14) in the context of Bayesian model fitting in §6.2.1.

[18] Alternatively we can also explicitly pick a target distribution over the output classes with multiple nonzero components $p(i|x_{\tilde{\alpha}})$, which is known as *soft labeling*. This can be implemented as a regularization technique called *label smoothing*, where $p(i^\star_{\tilde{\alpha}}|x_{\tilde{\alpha}}) = 1-\epsilon$ and $p(i \neq i^\star_{\tilde{\alpha}}|x_{\tilde{\alpha}}) = \epsilon/(n_{\text{out}}-1)$, or as *knowledge distillation*, mentioned in footnote 28 of §6, when you actually want to learn such a distribution over output classes.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

leading to the same softmax $q(i|x_\delta)$. Thus, we see explicitly that our solution (10.35) unambiguously describes networks fully-trained according to the cross-entropy loss.

10.2.4 Kernel Prediction

After an intelligence – artificial or otherwise – undergoes an intense memorization session, often that intelligence is then subjected to a *test* with unseen problems in order to probe its actual understanding. In the context of machine learning, we typically evaluate our model's understanding by asking it to make predictions on novel inputs $x_{\dot\beta}$ from the *test set* $\dot\beta \in \mathcal{B}$.

In the infinite-width limit, the predictions of a fully-trained MLP are governed by the stochastic equation

$$z^{(L)}_{i;\dot\beta}(T) = z^{(L)}_{i;\dot\beta} - \sum_{\tilde\alpha_1,\tilde\alpha_2\in\mathcal{A}} \Theta^{(L)}_{\dot\beta\tilde\alpha_1}\, \widetilde{\Theta}^{\tilde\alpha_1\tilde\alpha_2}_{(L)} \left(z^{(L)}_{i;\tilde\alpha_2} - y_{i;\tilde\alpha_2}\right), \tag{10.39}$$

whether we train the model in one step (10.28) or in many steps (10.35) – and regardless of the choice of loss function or any other details of the learning algorithm.[19] For clarity, note that the inverse frozen NTK $\widetilde{\Theta}^{\tilde\alpha_1\tilde\alpha_2}_{(L)}$ is taken with respect to the $N_\mathcal{A}$-by-$N_\mathcal{A}$ training-set submatrix only, while the frozen NTK $\Theta^{(L)}_{\dot\beta\tilde\alpha_1}$ is an *off-diagonal* block of the full frozen NTK, connecting an element of the training set to an element of the test set. Note also that the network outputs $z^{(L)}_{i;\delta}$ on the right-hand side of the equation are evaluated at initialization: once again, observables without any step argument should be assumed to be evaluated at initialization, while observables with a step argument T should be assumed to be evaluated at the end of training.

The stochastic equation (10.39) describes the predictions of a particular instantiation of a fully-trained neural network. The stochasticity arises from the fact that the prediction (10.39) depends on the network outputs at initialization $z^{(L)}_{i;\delta}$, which themselves depend on the particular realization of the initialized parameters $\theta_{\text{init}} \equiv \theta(t=0)$. Since we already know that such a network has completely memorized the training set so that $z^{(L)}_{i;\tilde\alpha}(T) = y_{i;\tilde\alpha}$, the stochasticity here means that any given network in the ensemble can potentially make different predictions on elements of the test set.

With that in mind, let us now compute the full distribution over such test-set predictions for our entire ensemble of fully-trained networks. Inspecting (10.39), we see that the prediction $z^{(L)}_{i;\dot\beta}(T)$ is a simple linear transformations of the Gaussian-distributed initial outputs $z^{(L)}_{i;\delta}$ and thus will itself be Gaussian. The mean prediction is simply given by

$$m^{\infty}_{i;\dot\beta} \equiv \mathbb{E}\left[z^{(L)}_{i;\dot\beta}(T)\right] = \sum_{\tilde\alpha_1,\tilde\alpha_2\in\mathcal{A}} \Theta^{(L)}_{\dot\beta\tilde\alpha_1}\, \widetilde{\Theta}^{\tilde\alpha_1\tilde\alpha_2}_{(L)} y_{i;\tilde\alpha_2}. \tag{10.40}$$

[19] Note that our analyses of §10.1 apply just as much to a small step as they do to a giant leap: the fully-trained infinite-width network has neither wiring in the vectorial components of the network output (§10.1.1) nor representation learning (§10.1.2).

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

This expression is entirely analogous to the infinite-width posterior mean prediction for exact Bayesian inference, (6.64), with a simple replacement of all types of frozen neural tangent kernels with kernels: $\Theta^{(L)} \to K^{(L)}$. (More on this soon.) Meanwhile, the covariance of the prediction (10.39) is given by

$$\begin{aligned}\text{Cov}\Big[z^{(L)}_{i_1;\dot\beta_1}(T),\, z^{(L)}_{i_2;\dot\beta_2}(T)\Big] &\equiv \mathbb{E}\left[z^{(L)}_{i_1;\dot\beta_1}(T)\, z^{(L)}_{i_2;\dot\beta_2}(T)\right] - m^{\infty}_{i_1;\dot\beta_1} m^{\infty}_{i_2;\dot\beta_2} \\ &= \delta_{i_1 i_2}\Bigg[K^{(L)}_{\dot\beta_1\dot\beta_2} - \sum_{\tilde\alpha_1,\tilde\alpha_2\in\mathcal{A}} \Theta^{(L)}_{\dot\beta_2\tilde\alpha_1}\widetilde{\Theta}^{\tilde\alpha_1\tilde\alpha_2}_{(L)} K^{(L)}_{\dot\beta_1\tilde\alpha_2} - \sum_{\tilde\alpha_1,\tilde\alpha_2\in\mathcal{A}} \Theta^{(L)}_{\dot\beta_1\tilde\alpha_1}\widetilde{\Theta}^{\tilde\alpha_1\tilde\alpha_2}_{(L)} K^{(L)}_{\dot\beta_2\tilde\alpha_2} \\ &\quad + \sum_{\tilde\alpha_1,\tilde\alpha_2,\tilde\alpha_3,\tilde\alpha_4\in\mathcal{A}} \Theta^{(L)}_{\dot\beta_1\tilde\alpha_1}\widetilde{\Theta}^{\tilde\alpha_1\tilde\alpha_2}_{(L)}\Theta^{(L)}_{\dot\beta_2\tilde\alpha_3}\widetilde{\Theta}^{\tilde\alpha_3\tilde\alpha_4}_{(L)} K^{(L)}_{\tilde\alpha_4\tilde\alpha_2}\Bigg]. \end{aligned} \tag{10.41}$$

While this expression is somewhat complicated looking, involving both kernels and frozen NTKs, it similarly reduces to the Bayesian infinite-width posterior covariance (6.57) with the substitution $\Theta^{(L)} \to K^{(L)}$.[20]

This ensemble of network predictions (10.39), completely specified by its mean (10.40) and covariance (10.41), defines a kind of **generalized posterior distribution**. This distribution comprises a complete closed-form solution for our ensemble of infinite-width networks at the end of training, regardless of the path that we take to get there. The mean of the distribution is the prediction of the network *averaged* over instantiations, while the covariance quantifies the instantiation-to-instantiation fluctuations of these predictions.

Indeed, it is sensible to identify the ensemble of trained networks as a kind of posterior distribution, if you recall our discussion of approximation methods for Bayesian inference in §6.2.1: minimizing a training loss $\mathcal{L}_{\mathcal{A}}(\theta)$ gives the maximum likelihood estimation (MLE) of the model parameters (6.21), which we now identify with our fully-trained solution (10.32) $\theta^{\star}_{\text{MLE}} = \theta^{\star}_{\text{Newton}}$. Further recalling the content of footnote 11 in §6.2.1, for wide networks the minimum of the loss is not unique, and the MLE approach will give a family of minima parameterized by the initialization: $\theta^{\star}_{\text{MLE}}(\theta_{\text{init}})$.[21] This lack

[20] The fact that exact Bayesian inference and gradient descent in general make different predictions is indicative of the fact that – for general hyperparameter settings – they are actually different learning algorithms.

[21] As per that same footnote, we could also try to analyze the MAP estimate (6.22) in the context of infinite-width gradient-based learning with the addition of a regularization term of the form $\sum_{\mu=1}^{P} a_\mu \theta_\mu^2$ to the loss. If you start this analysis, you'll immediately find that the gradient-descent update $d\theta_\mu$ includes an additional term $-2\eta \sum_\nu \lambda_{\mu\nu}\, a_\nu \theta_\nu$, and after some reflection you'll likely also realize the need to define a new stochastic tensor,

$$\widehat{\mathcal{R}}^{(\ell)}_{i;\delta} \equiv \sum_{\mu,\nu} \lambda_{\mu\nu}\, a_\mu \theta_\mu \frac{dz^{(\ell)}_{i;\delta}}{d\theta_\nu}, \tag{10.42}$$

which has a stochastic iteration given by

$$\widehat{\mathcal{R}}^{(\ell+1)}_{i;\delta} = a_b^{(\ell+1)}\lambda_b^{(\ell+1)} b_i^{(\ell+1)} + a_W^{(\ell+1)}\frac{\lambda_W^{(\ell+1)}}{n_\ell}\sum_{j=1}^{n_\ell} W_{ij}^{(\ell+1)}\sigma^{(\ell)}_{j;\delta} + \sum_{j=1}^{n_\ell} W_{ij}^{(\ell+1)}\sigma'^{(\ell)}_{j;\delta}\widehat{\mathcal{R}}^{(\ell)}_{j;\delta}, \tag{10.43}$$

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

of uniqueness ultimately stems from the lingering dependence of the trained network prediction $z^{(L)}_{i;\dot{\beta}}(T)$ on the initial function output $z^{(L)}_{i;\delta}$, which stochastically varies from instantiation to instantiation, cf. our prediction (10.39). Considering the ensemble over instantiations of θ_{init}, we now see exactly how this generalized distribution with mean (10.40) and covariance (10.41) depends on the training hyperparameters $\lambda_b^{(\ell)}$ and $\lambda_W^{(\ell)}$, initialization hyperparameters $C_b^{(\ell)}$ and $C_W^{(\ell)}$, and training data $(x_{\tilde{\alpha}}, y_{\tilde{\alpha}})_{\tilde{\alpha}\in\mathcal{A}}$.

To be a little pedantic for a paragraph, the covariance in the generalized posterior distribution (10.41) really has a different interpretation than the posterior covariance (6.57) we computed for exact Bayesian inference at infinite width. In the setting of gradient-based learning, the covariance of the output encodes the variation in the predictions among networks in the ensemble, each corresponding to different parameter settings that still minimize the training loss $\mathcal{L}_{\mathcal{A}}(\theta)$. In the setting of exact Bayesian inference, the covariance encodes our intrinsic uncertainty about unseen data, and a small uncertainty can serve as a measure of confidence in our prediction. Thus, these covariances arise for different reasons and are epistemologically quite different in nature.

However, if we can be somewhat pragmatic for a sentence, when you have multiple trained models, it's not entirely unreasonable to try to think of this generalized posterior covariance (10.41) as a measure of confidence as well.

That One Place Where Gradient Descent = Exact Bayesian Inference

Just before, we casually noticed that if we replaced all frozen *neural tangent kernels* with *kernels*, $\Theta^{(L)} \to K^{(L)}$, then the generalized posterior distribution based on (10.39) reduces to the exact Bayesian posterior distribution (6.66). Let us now show how we can actually implement such a substitution in the context of gradient-based learning with a particular choice of training hyperparameters.

To see how to do this, let's put side-by-side the recursion that defines the output-layer kernel (4.118) and the recursion that defines the output-layer frozen NTK (9.5):

$$K^{(L)}_{\delta_1\delta_2} = C_b^{(L)} + C_W^{(L)} \left\langle \sigma_{\delta_1}\sigma_{\delta_2} \right\rangle_{K^{(L-1)}} , \tag{10.44}$$

$$\Theta^{(L)}_{\delta_1\delta_2} = \lambda_b^{(L)} + \lambda_W^{(L)} \left\langle \sigma_{\delta_1}\sigma_{\delta_2} \right\rangle_{K^{(L-1)}} + C_W^{(L)} \left\langle \sigma'_{\delta_1}\sigma'_{\delta_2} \right\rangle_{K^{(L-1)}} \Theta^{(L-1)}_{\delta_1\delta_2}. \tag{10.45}$$

By inspection, it's immediately clear that setting the final-layer learning rates as

$$\lambda_b^{(L)} = C_b^{(L)}, \qquad \lambda_W^{(L)} = C_W^{(L)} \tag{10.46}$$

gives us what we want, almost:

$$\Theta^{(L)}_{\delta_1\delta_2} = K^{(L)}_{\delta_1\delta_2} + C_W^{(L)} \left\langle \sigma'_{\delta_1}\sigma'_{\delta_2} \right\rangle_{K^{(L-1)}} \Theta^{(L-1)}_{\delta_1\delta_2}. \tag{10.47}$$

---

and whose cross-correlation with the preactivations you'll want to compute. Here, you will have defined separate layer-dependent bias and weight regularizations for the coefficients a_μ analogous to what we did for the learning-rate tensor in (8.5), and you may want to work out the interplay between these regularization hyperpameters and initialization hyperparameters for extra credit.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

To get rid of that pesky last term, we need a way to make the penultimate-layer frozen NTK $\Theta^{(L-1)}_{\delta_1\delta_2}$vanish. To ensure this, we can simply set all the other training hyperparameters to zero:

$$\lambda_b^{(\ell)} = 0\,, \qquad \lambda_W^{(\ell)} = 0\,, \qquad \text{for} \quad \ell < L. \tag{10.48}$$

Combined with the initial condition for the NTK recursion (8.23), this ensures that $\Theta^{(\ell)}_{\delta_1\delta_2} = 0$ for $\ell < L$, including $\ell = L-1$. Hence, this particular configuration of training hyperparameters sets

$$\Theta^{(L)}_{\delta_1\delta_2} = K^{(L)}_{\delta_1\delta_2}, \tag{10.49}$$

giving us what we wanted, exactly.

In words, this choice of the training hyperparameters (10.46) and (10.48) means that we are training only the biases and weights in the last layer. This establishes that, in the infinite-width limit, exact Bayesian inference is actually a very special case of an ensemble of networks trained with gradient-based learning.

In practice, this means that one could train an ensemble of networks with gradient descent using the training hyperparameter choices (10.46) and (10.48) in order to implement a very good approximation of exact Bayesian inference. (The approximation would become exact if you had an infinite number of networks in your ensemble.) On the one hand, unlike the exact version of Bayesian inference presented in §6.3, in this case we no longer need to explicitly store or invert the kernel. On the other hand, we may need to fully train a large number of very wide networks for our ensemble to give a good approximation, which may again be expensive in terms of computation and memory.

Interestingly, by explicitly turning off the learning in the hidden layers, we are significantly changing the features used to compute our predictions. In particular, in this case the NTK only has contributions from the biases and weights in the final layer. Intuitively, what's happening here is that we're taking random features in the penultimate layer $\sigma\left(z_i^{(L-1)}\right)$ and then explicitly training the biases $b^{(L)}$ and weights $W^{(L)}$ to fit the best possible *linear model* of these random features.[22]

10.3 Generalization

As remarked before, ultimately we care about how well a model performs on a previously unseen test set $\mathcal{B}$ as compared to the training set $\mathcal{A}$. Some fully-trained networks will *generalize* to these new examples better than others, depending strongly on their initialization and training hyperparameters.

To assess this, we can compute the **generalization error**:

$$\mathcal{E} \equiv \mathcal{L}_{\mathcal{B}} - \mathcal{L}_{\mathcal{A}}. \tag{10.50}$$

[22] We'll explain in §10.4 that the general version of gradient-based learning in the infinite-width limit is also a *linear model* of random features but is constructed from a larger set of such features encompassing all the hidden layers.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

The generalization error is a quantitative measure of how well a network is really approximating the desired function.[23] Specifically, if the training loss is small but the test loss is large, such that there's significant generalization error, then the network isn't really a good model of $f(x)$; instead it's just a lookup table of the values of $f(x)$ when x is taken from the training set $\mathcal{A}$. This is known as **overfitting**. In contrast, if the training and test losses are both small such that there's little generalization error, then we expect that our model is going beyond simple memorization.[24] As such, the generalization error is often considered to be the main quantitative measure of success of a machine learning model.

In the infinite-width limit, we saw in the last section that we can easily set the training loss to zero $\mathcal{L}_{\mathcal{A}} = 0$ for any particular network. Thus, in the current context the generalization error is completely assessed by the test loss,

$$\mathcal{E} = \mathcal{L}_{\mathcal{B}}, \tag{10.51}$$

and our current goal is to understand the statistics of the test error in order to characterize how infinite-width networks generalize.

For wide networks we know that there are many configurations of the model parameters that will minimize the training loss and memorize the training data. Some of these configurations might generalize well leading to a small test loss, while some might overfit leading to a large test loss. Thus, the statistics of the generalization error are determined by the statistics of these configurations, which are in turn determined by our initialization and training hyperparameters.

The mean generalization error captures the generalization properties of the ensemble of networks, and its variance tells us how the instantiation-to-instantiation fluctuations lead some particular networks to generalize better than others. Understanding these statistics will inform how we should pick our hyperparameters to achieve the best generalization performance on average as well as to ensure that the typical behavior of any fully-trained network is likely to be close to the average.[25]

With that in mind, let's first evaluate the MSE test loss, averaged over an ensemble of fully-trained networks:

[23] Without loss of generality, in the following discussion we'll implicitly assume that the minimal value of the loss is zero.

[24] If the generalization error $\mathcal{E}$ is small but the training loss $\mathcal{L}_{\mathcal{A}}$ is large, then the model is said to be *underfitting*. This situation is not really relevant for very wide networks since, as we've already explained, we can fully train them to achieve zero training loss.

[25] In some applications of machine learning, unseen examples are further divided into two types of datasets, *(i)* a *validation set*, used generally for model selection and often specifically for tuning hyperparameters, and *(ii)* a *test set*, used to assess the generalization properties of a particular trained model. Despite our liberal usage of the term *test set*, here, as we analytically compute the statistics of the loss on unseen examples and then use them to tune the hyperparameters, what we have is really closer in meaning to a *validation set*, as we are tuning an ensemble of models rather than assessing any particular one.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

$$\begin{aligned}\mathbb{E}\left[\mathcal{L}_{\mathcal{B}}(T)\right] &= \mathbb{E}\left[\frac{1}{2}\sum_{i=1}^{n_L}\sum_{\dot{\beta}\in\mathcal{B}}\left(z^{(L)}_{i;\dot{\beta}}(T) - y_{i;\dot{\beta}}\right)^2\right] \\ &= \mathbb{E}\left[\frac{1}{2}\sum_{i=1}^{n_L}\sum_{\dot{\beta}\in\mathcal{B}}\left(z^{(L)}_{i;\dot{\beta}}(T) - m^{\infty}_{i;\dot{\beta}} + m^{\infty}_{i;\dot{\beta}} - y_{i;\dot{\beta}}\right)^2\right] \\ &= \frac{1}{2}\sum_{\dot{\beta}\in\mathcal{B}}\left\{\sum_{i=1}^{n_L}\left(m^{\infty}_{i;\dot{\beta}} - y_{i;\dot{\beta}}\right)^2 + \sum_{i=1}^{n_L}\mathrm{Cov}\left[z^{(L)}_{i;\dot{\beta}}(T),\, z^{(L)}_{i;\dot{\beta}}(T)\right]\right\}. \end{aligned} \tag{10.52}$$

In the second line, we added and subtracted the infinite-width mean prediction (10.40), and to get to the third line we noted that the cross terms cancel under the expectation.

This decomposition (10.52) illustrates a type of generalized **bias–variance tradeoff**: the first term, the *bias*, measures the deviation of the mean prediction of the ensemble $m^{\infty}_{i;\dot{\beta}}$ from the true output $y_{i;\dot{\beta}}$; the second term, the *variance* – or specifically the covariance of the generalized posterior distribution (10.41) – measures the instantiation-to-instantiation fluctuations of that prediction across different models in our ensemble. The reason why this is called a *tradeoff* is that typically different settings of the hyperparameters will decrease one term at the cost of increasing the other, requiring the modeler to choose whether to improve one at the cost of the other.[26]

For more general losses, we can Taylor-expand the test loss around the mean prediction as

$$\begin{aligned}\mathcal{L}_{\mathcal{B}} = \mathcal{L}_{\mathcal{B}}(m^{\infty}) &+ \sum_{i,\dot{\beta}}\left.\frac{\partial\mathcal{L}_{\mathcal{B}}}{\partial z^{(L)}_{i;\dot{\beta}}}\right|_{z^{(L)}=m^{\infty}}\left(z^{(L)}_{i;\dot{\beta}}(T) - m^{\infty}_{i;\dot{\beta}}\right) \\ &+ \frac{1}{2}\sum_{i_1,i_2,\dot{\beta}_1,\dot{\beta}_2}\left.\frac{\partial^2\mathcal{L}_{\mathcal{B}}}{\partial z^{(L)}_{i_1;\dot{\beta}_1}\partial z^{(L)}_{i_2;\dot{\beta}_2}}\right|_{z^{(L)}=m^{\infty}}\left(z^{(L)}_{i_1;\dot{\beta}_1}(T) - m^{\infty}_{i_1;\dot{\beta}_1}\right)\left(z^{(L)}_{i_2;\dot{\beta}_2}(T) - m^{\infty}_{i_2;\dot{\beta}_2}\right) + \cdots, \end{aligned} \tag{10.53}$$

where we've denoted the test loss evaluated at the mean prediction as

$$\mathcal{L}_{\mathcal{B}}(m^{\infty}) \equiv \mathcal{L}_{\mathcal{B}}\left(z^{(L)} = m^{\infty}\right). \tag{10.54}$$

Performing the expectation over the ensemble and noting that $\mathbb{E}\left[z^{(L)}_{i;\dot{\beta}}(T) - m^{\infty}_{i;\dot{\beta}}\right] = 0$ by definition (10.40), we get

$$\mathbb{E}\left[\mathcal{L}_{\mathcal{B}}\right] = \mathcal{L}_{\mathcal{B}}(m^{\infty}) + \frac{1}{2}\sum_{i_1,i_2,\dot{\beta}_1,\dot{\beta}_2}\left.\frac{\partial^2\mathcal{L}_{\mathcal{B}}}{\partial z^{(L)}_{i_1;\dot{\beta}_1}\partial z^{(L)}_{i_2;\dot{\beta}_2}}\right|_{z^{(L)}=m^{\infty}}\mathrm{Cov}\left[z^{(L)}_{i_1;\dot{\beta}_1}(T),\, z^{(L)}_{i_2;\dot{\beta}_2}(T)\right] + \cdots. \tag{10.55}$$

[26] The reason why we call it *generalized* bias–variance tradeoff is that, in the *standard* bias–variance tradeoff, the expectation is over different realizations of the training set $\mathcal{A}$ rather than over different initializations of the model parameters θ_μ. In that typical setting we have only a single model, and the *bias* characterizes how well that model can be trained on each different training set – with a large bias indicative of *underfitting* – while the *variance* characterizes the fluctuations of that model's performance over the different training sets – with a large variance indicative of *overfitting*.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Here again we find a generalized bias–variance decomposition: the first term $\mathcal{L}_{\mathcal{B}}(m^{\infty})$ is the bias, measuring the deviation of the mean prediction from the true output on the test set, and the second term is the variance, measuring the instantiation-to-instantiation uncertainty as the trace of the covariance multiplied by the Hessian of the loss with respect to the network outputs. Thus, for *any* choice of loss function, these bias and variance terms will give a good proxy for the generalization error $\mathcal{E}$ so long as our models are making predictions that are close to the mean prediction.

Now, let's see how to compute these bias and variance terms in a few different setups. In most common cases in practice, the loss is *extensive* or additive in samples, i.e., $\mathcal{L}_{\mathcal{B}} = \sum_{\dot{\beta}\in\mathcal{B}} \mathcal{L}_{\dot{\beta}}$, and we can consider the test loss evaluated on one test sample at a time. Thus, for the purpose of our analysis here, the question is: for a given test input, how many training examples are relevant for making a prediction?

In §10.3.1, we'll compute the bias and variance terms of the generalization error (10.55) around one training sample using our δ expansion introduced in §5, giving another lens into hyperparameter tuning and criticality for our two universality classes. However, this view will be somewhat limited by the restriction of our training set to one sample.

In §10.3.2, we'll enlarge our training set to include two samples. Rather than computing the generalization error itself, here we'll be able to explore directly a different aspect of generalization: how a fully-trained network either interpolates or extrapolates to make predictions.

10.3.1 Bias–Variance Tradeoff and Criticality

Let us index one training sample by $\tilde{\alpha} = +$ and a nearby test sample by $\dot{\beta} = -$; let us also focus on the output layer $\ell = L$ and temporarily drop the layer index from the frozen NTK, $\Theta_{\delta_1\delta_2} \equiv \Theta^{(L)}_{\delta_1\delta_2}$, until later when we need to discuss the depth dependence of various NTK components.

The bias term in the generalization error is determined by the deviation of the mean prediction (10.40) from the true output:

$$\begin{aligned} m^{\infty}_{i;-} - y_{i;-} &= \frac{\Theta_{-+}}{\Theta_{++}} y_{i;+} - y_{i;-} \\ &= (y_{i;+} - y_{i;-}) + \left(\frac{\Theta_{-+}}{\Theta_{++}} - 1\right) y_{i;+}. \end{aligned} \tag{10.56}$$

In this expression, the first term is the true difference in the function outputs on the two different inputs, $f(x_+) - f(x_-)$, while the second term is a similar (expected) difference between our predicted output on x_- and the learned true output on x_+, $z_-(T) - z_+(T)$.[27] Note the opposite ordering of $+$ and $-$ in these two terms: if our prediction is exactly

[27] In general, the bias term $\mathcal{L}_{\mathcal{B}}(m^{\infty})$ in the generalization error (10.55) depends on the details of the loss. Of course, we can expand $\mathcal{L}_{\mathcal{B}}(m^{\infty})$ around the true output $y_{i;-}$, and the expansion will depend on the difference $m^{\infty}_{i;-} - y_{i;-}$ (10.56). For the MSE loss, the bias is precisely the square of this difference. For the cross-entropy loss, it is more natural to expand in terms of the difference $\overline{q}(i|x_-) - p(i|x_-)$, with $\overline{q}(i|x_\delta) \equiv \exp\left[m^{\infty}_{i;\delta}\right] / \sum_{j=1}^{n_{\text{out}}} \exp\left[m^{\infty}_{j;\delta}\right]$.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

correct, these two terms are equal in magnitude and opposite in sign, and the bias term in the generalization error will vanish.

With that in mind, the quantity in parentheses in the second factor of the bias term (10.56),

$$\frac{\Theta_{-+}}{\Theta_{++}} - 1, \tag{10.57}$$

serves as a natural measure of *robustness* since it characterizes how sensitively our prediction changes – i.e., how $|z_-(T) - z_+(T)|$ grows – with corresponding small changes in the input. A model that isn't robust will often be incorrect, making predictions that vary greatly from the network output on nearby training points, while too robust a model will not have much flexibility in its output. Since a priori we don't know what type of function we are going to approximate, we naturally would want to pick hyperparameters that include a class of networks that are robust but not overly inflexible.[28]

Now, since we're considering a test input that's nearby our training input, that should remind you of our δ expansion from our criticality analysis in §5.1.[29] Specifically, we can expand the frozen NTK in our $\gamma^{[a]}$ basis as we did for the kernel in (5.15),

$$\Theta_{\pm\pm} = \Theta_{[0]} \pm \Theta_{[1]} + \Theta_{[2]}\,, \qquad \Theta_{\pm\mp} = \Theta_{[0]} - \Theta_{[2]}, \tag{10.58}$$

and make δ expansions similar to the ones we did for the kernel in (5.22)–(5.24),

$$\Theta_{[0]} = \Theta_{00} + \delta\delta\Theta_{[0]} + O\left(\delta^4\right), \tag{10.59}$$

$$\Theta_{[1]} = \delta\Theta_{[1]} + O\left(\delta^3\right), \tag{10.60}$$

$$\Theta_{[2]} = \delta\delta\Theta_{[2]} + O\left(\delta^4\right), \tag{10.61}$$

where the expansion is taken around the midpoint frozen NTK Θ_{00} evaluated on the midpoint input $x_{i;0} \equiv (x_{i;+} + x_{i;-})/2$.

For simplicity of our presentation, let's now assume that the two inputs have the same norm $\sum_{i=1}^{n_0} x_{i;+}^2 = \sum_{i=1}^{n_0} x_{i;-}^2$, so that $K_{[1]} = 0$ and $\Theta_{[1]} = (\Theta_{++} - \Theta_{--})/2 = 0$. With this simplification, plugging the decomposition (10.58) and then the expansions (10.59) and (10.61) into the expression for our robustness measure (10.57), we get

$$\frac{\Theta_{-+}}{\Theta_{++}} - 1 = \left(\frac{\Theta_{[0]} - \Theta_{[2]}}{\Theta_{[0]} + \Theta_{[2]}} - 1\right) = -2\frac{\delta\delta\Theta_{[2]}}{\Theta_{00}} + O\left(\delta^4\right). \tag{10.62}$$

Thus, we see that the ratio $\delta\delta\Theta_{[2]}/\Theta_{00}$ captures the robustness of predictions for nearby test inputs. We'll analyze its depth dependence for two universality classes shortly.

[28] A dedicated reader might notice the parallel with §6.3.1, where we argued for criticality by considering the evidence for two inputs with differing true outputs, $f(x_+) - f(x_-) \neq 0$, which called for similar flexibility in choice of function approximators.

[29] The following applies to smooth activation functions and is intended to give the general picture. We will give an analysis particular to nonlinear scale-invariant activation functions later when discussing them in particular.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Having covered the bias term in the generalization error, let's next consider the variance term. The loss-independent piece of the variance is given by the covariance of the generalized posterior distribution (10.41). Evaluating (10.41) for a single training sample $\tilde{\alpha} = +$, using our decompositions for the kernel (5.15) and frozen NTK (10.58), and then using expansions (5.22), (5.24), (10.59), and (10.61), we find

$$\begin{aligned}
&\operatorname{Cov}\left[z^{(L)}_{i;-}(T),\, z^{(L)}_{i;-}(T)\right] \qquad (10.63)\\
&\quad = K_{--} - 2\frac{\Theta_{-+}}{\Theta_{++}}K_{-+} + \left(\frac{\Theta_{-+}}{\Theta_{++}}\right)^2 K_{++}\\
&\quad = K_{[0]} + K_{[2]} - 2\left(\frac{\Theta_{[0]} - \Theta_{[2]}}{\Theta_{[0]} + \Theta_{[2]}}\right)\left(K_{[0]} - K_{[2]}\right) + \left(\frac{\Theta_{[0]} - \Theta_{[2]}}{\Theta_{[0]} + \Theta_{[2]}}\right)^2\left(K_{[0]} + K_{[2]}\right)\\
&\quad = 4\delta\delta K_{[2]} + O\left(\delta^4\right).
\end{aligned}$$

Thus, to leading order, the variance term depends only on the perpendicular perturbation of the kernel $\delta\delta K_{[2]}$.

At this point, we know everything there is to know about how $\delta\delta K_{[2]}$ behaves as a function of depth for our universality classes (cf. §5.3 and §5.5). On the one hand, we could pick initialization hyperparameters such that $\delta\delta K_{[2]}$ grows exponentially with depth. However, with this choice the variance term will grow very quickly, leading to large fluctuations in model predictions between different realizations. On the other hand, we could pick initialization hyperparameters that decay exponentially with depth, leading to a quickly vanishing variance term and very overconfident predictions. However, we will soon see that this overconfidence comes at a cost: an exponentially vanishing perpendicular perturbation $\delta\delta K_{[2]}$ implies an exponentially vanishing frozen NTK component $\delta\delta\Theta_{[2]}$ and thus a vanishing robustness measure (10.62), signaling extreme inflexibility. In particular, we will have learned a constant function that's always equal to y_+, regardless of the input.

This is precisely the generalized bias–variance tradeoff that we described above: if we try to set the variance to zero by having $\delta\delta K_{[2]}$ vanish exponentially, then the vanishing of $\delta\delta\Theta_{[2]}$ will cause our bias to be larger for generic inputs, and consequently the network will not be able to generalize in a nontrivial manner. Vice versa, making the function too flexible with large $\delta\delta\Theta_{[2]}$ will not only make the model predictions too sensitive to small changes in the input through $\delta\delta\Theta_{[2]}$ but also will cause large fluctuations in that prediction from realization to realization through $\delta\delta K_{[2]}$.

Of course, we know that there's a third option: we could pick our criticality condition $\chi_\perp(K^\star) = 1$. This setting of the initialization hyperparameters has the potential to balance the bias–variance tradeoff, leading to the best outcome without a priori knowing anything more about the underlying dataset we're trying to model.

What about our other criticality condition, $\chi_\parallel(K^\star) = 1$? Recall from our discussion of the exploding and vanishing gradient problem in §9.4 that the parallel susceptibility $\chi_\parallel$ affects the way in which the midpoint frozen NTK Θ_{00} receives contributions from different layers. As the midpoint frozen NTK Θ_{00} appears in the robustness measure as in (10.62), this suggests that it also plays an important role in generalization. In fact,

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

we will see soon in §10.4 that ensuring equal contributions from all layers is another way of saying that we use the greatest set of features available to us in making a prediction. Thus, it stands to reason that also picking the criticality condition $\chi_{\parallel}(K^\star) = 1$ in conjunction with the condition $\chi_{\perp}(K^\star) = 1$ is a natural choice for generalization, in addition to all our other evidence for such a choice.[30]

Now, returning to the bias part of the generalization error, to complete our analysis we'll need to solve a recursion for the $\delta\delta\Theta_{[2]}$ component of the frozen NTK recursion (9.5), reprinted here in full:

$$\Theta^{(\ell+1)}_{\delta_1\delta_2} = \lambda_b^{(\ell+1)} + \lambda_W^{(\ell+1)} \left\langle \sigma_{\delta_1}\sigma_{\delta_2} \right\rangle_{K^{(\ell)}} + C_W \left\langle \sigma'_{\delta_1}\sigma'_{\delta_2} \right\rangle_{K^{(\ell)}} \Theta^{(\ell)}_{\delta_1\delta_2}. \tag{10.64}$$

Let's first work this out for the $K^\star = 0$ universality class, and then we'll consider the scale-invariant universality class, for which we'll need to make use of our finite-angle results from §5.5. Either way, this should be child's play for us at this point.[31]

[30] Just like in footnote 23 of §6.3.1, additional justification comes from the consideration of two inputs with unequal norms: $\sum_{i=1}^{n_0} x_{i;+}^2 \neq \sum_{i=1}^{n_0} x_{i;-}^2$. In such a case, the robustness measure (10.62) is given by

$$\frac{\Theta_{-+}}{\Theta_{++}} - 1 = -\frac{\delta\Theta_{[1]}}{\Theta_{00}} - 2\frac{\delta\delta\Theta_{[2]}}{\Theta_{00}} + \left(\frac{\delta\Theta_{[1]}}{\Theta_{00}}\right)^2 + O\left(\delta^3\right), \tag{10.65}$$

and the covariance is given by

$$\mathrm{Cov}\left[z^{(L)}_{i;-}(T),\, z^{(L)}_{i;-}(T)\right] = 4\delta\delta K_{[2]} - 2\delta K_{[1]}\frac{\delta\Theta_{[1]}}{\Theta_{00}} + K_{00}\left(\frac{\delta\Theta_{[1]}}{\Theta_{00}}\right)^2 + O\left(\delta^3\right). \tag{10.66}$$

First, we see that the kernel components $\delta K_{[1]}$ and K_{00} both contribute, necessitating that we set $\chi_{\parallel} = 1$ as per our previous discussions. In addition, we will also need to tame the exploding and vanishing problem of $\delta\Theta_{[1]}$.

For the scale-invariant universality class, $\Theta_{[1]} = (\Theta_{++} - \Theta_{--})/2$ has exactly the same depth dependence as the single-input frozen NTK (9.44). In this case, $\chi_{\parallel} = \chi_{\perp} \equiv \chi$, and all the exponential explosions and vanishments are mitigated by setting $\chi = 1$.

For the $K^\star = 0$ universality class, we can write a recursion for $\delta\Theta_{[1]}$ by projecting out the $\gamma^{[1]}$ component of the full frozen NTK recursion (10.64) using (5.20):

$$\delta\Theta^{(\ell+1)}_{[1]} = \chi_{\perp}^{(\ell)}\delta\Theta^{(\ell)}_{[1]} + \left(\frac{\lambda_W^{(\ell+1)}}{C_W}\chi_{\parallel}^{(\ell)} + \frac{C_W}{K_{00}^{(\ell)}}\left\langle z\sigma'\sigma'' \right\rangle_{K_{00}^{(\ell)}}\Theta_{00}^{(\ell)}\right)\delta K^{(\ell)}_{[1]}, \tag{10.67}$$

i.e., with a derivation almost isomorphic to the one below for $\delta\delta\Theta_{[2]}$ (10.70). We in particular see that $\delta K^{(\ell)}_{[1]}$ contributes to $\delta\Theta^{(\ell)}_{[1]}$ – which can be thought of as the Bayesian contribution per our last discussion in §10.2.4 – and its exploding and vanishing problem is mitigated by setting $\chi_{\parallel}(K^\star) = 1$: cf. (5.47). (At this point you may find it useful to re-read and re-flect on the last paragraph of footnote 23 in §6.3.1.) You can further study the depth dependence of $\delta\Theta^{(\ell)}_{[1]}$ at criticality and find that $\delta\Theta^{(\ell)}_{[1]}$ decays faster than $\Theta^{(\ell)}_{00}$ and $\delta\delta\Theta^{(\ell)}_{[2]}$, thus reducing the problem back to the one studied in the main text.

[31] An even more childish play would be studying the Bayesian version of generalization error by setting the training hyperparameters according to (10.46) and (10.48), such that $\Theta^{(L)}_{\delta_1\delta_2} = K^{(L)}_{\delta_1\delta_2}$. In this case, we know exactly how the bias and variance terms of the generalization error behave. This is a very particular setting of the training hyperparameters and unlikely to be optimal in general (cf. our discussion of the differences between the frozen NTK and Bayesian kernel in terms of feature functions in §10.4). Indeed for the scale-invariant universality class, we'll explicitly see in footnote 37 that exact Bayesian inference has inferior asymptotic behavior compared to the more general gradient-based learning.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

$K^\star = 0$ Universality Class

Recall (5.44) from much much earlier describing the decomposition of the Gaussian expectation of two activations in the $\gamma^{[a]}$ basis. With the parallel perturbation turned off, $K^{(\ell)}_{[1]} = 0$, this expansion reads

$$\langle \sigma_{\delta_1} \sigma_{\delta_2} \rangle_{K^{(\ell)}} = \left[\langle \sigma\sigma \rangle_{K^{(\ell)}_{00}} + O\left(\delta^2\right) \right] \gamma^{[0]}_{\delta_1\delta_2} + \left[\delta\delta K^{(\ell)}_{[2]} \langle \sigma'\sigma' \rangle_{K^{(\ell)}_{00}} + O\left(\delta^4\right) \right] \gamma^{[2]}_{\delta_1\delta_2}. \tag{10.68}$$

With a replacement $\sigma \to \sigma'$, we have a similar decomposition for the Gaussian expectation of the derivatives of activations:

$$\langle \sigma'_{\delta_1} \sigma'_{\delta_2} \rangle_{K^{(\ell)}} = \left[\langle \sigma'\sigma' \rangle_{K^{(\ell)}_{00}} + O\left(\delta^2\right) \right] \gamma^{[0]}_{\delta_1\delta_2} + \left[\delta\delta K^{(\ell)}_{[2]} \langle \sigma''\sigma'' \rangle_{K^{(\ell)}_{00}} + O\left(\delta^4\right) \right] \gamma^{[2]}_{\delta_1\delta_2}. \tag{10.69}$$

Plugging these expansions into the full frozen NTK recursion (10.64) and using the component-wise identities $\gamma^{[0]}_{\delta_1\delta_2}\gamma^{[0]}_{\delta_1\delta_2} = \gamma^{[2]}_{\delta_1\delta_2}\gamma^{[2]}_{\delta_1\delta_2} = \gamma^{[0]}_{\delta_1\delta_2}$ and $\gamma^{[0]}_{\delta_1\delta_2}\gamma^{[2]}_{\delta_1\delta_2} = \gamma^{[2]}_{\delta_1\delta_2}$, we get

$$\delta\delta\Theta^{(\ell+1)}_{[2]} = \chi^{(\ell)}_\perp \delta\delta\Theta^{(\ell)}_{[2]} + \left(\frac{\lambda^{(\ell+1)}_W}{C_W} \chi^{(\ell)}_\perp + C_W \langle \sigma''\sigma'' \rangle_{K^{(\ell)}_{00}} \Theta^{(\ell)}_{00} \right) \delta\delta K^{(\ell)}_{[2]}, \tag{10.70}$$

where we've recalled the definition of the perpendicular susceptibility (5.51), $\chi^{(\ell)}_\perp = C_W \langle \sigma'\sigma' \rangle_{K^{(\ell)}_{00}}$.[32]

We learned long ago that the perpendicular susceptibility governs the behavior of the perpendicular perturbation $\delta\delta K^{(\ell)}_{[2]}$, and we see from (10.70) that it also controls the behavior of the frozen NTK component $\delta\delta\Theta^{(\ell)}_{[2]}$. As we alluded to before, the exponential decay/growth of $\delta\delta K^{(\ell)}_{[2]}$ and $\delta\delta\Theta^{(\ell)}_{[2]}$ are thusly linked. In particular, trying to eliminate the variance term of the generalization error by letting $\delta\delta K^{(\ell)}_{[2]}$ exponentially decay will also cause $\delta\delta\Theta^{(\ell)}_{[2]}$ to exponentially decay, making the model prediction constant, inflexible, and highly biased.

Given this and our previous discussion on the role of the parallel susceptibility $\chi_\parallel$, let's now tune to criticality, $\chi_\parallel(K^\star) = \chi_\perp(K^\star) = 1$, and evaluate the depth dependence of $\delta\delta\Theta^{(\ell)}_{[2]}$. For the $K^\star = 0$ universality class, criticality (5.90) is found by tuning $C_b = 0$ and $C_W = \frac{1}{\sigma_1^2}$. With these settings, we recall the large-ℓ asymptotic solutions from (5.92) and (5.99),

$$K^{(\ell)}_{00} = \left[\frac{1}{(-a_1)} \right] \frac{1}{\ell} + \cdots, \qquad \delta\delta K^{(\ell)}_{[2]} = \frac{\delta^2}{\ell^{p_\perp}} + \cdots, \tag{10.71}$$

where $p_\perp \equiv b_1/a_1$, the activation-function-dependent constants a_1 and b_1 were defined in (5.86) and (5.88), and δ^2 is a constant related to the initial separation of the inputs

[32] More generally there are terms proportional to $\left(\delta K_{[1]}\right)^2$ and $\delta K_{[1]}\delta\Theta_{[1]}$ in this recursion for $\delta\delta\Theta_{[2]}$; however, when training and test inputs have equal norms, $\delta K_{[1]} = 0$, these terms vanish.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

but isn't fixed by the asymptotic analysis. Also recall from the more recent past (9.76) that we can asymptotically expand the perpendicular susceptibility as

$$\chi_{\perp}^{(\ell)} = 1 - \frac{p_{\perp}}{\ell} + \cdots . \tag{10.72}$$

Similarly, by a simple Gaussian integral we can evaluate the following Gaussian expectation:

$$\left\langle \sigma''\sigma'' \right\rangle_{K_{00}^{(\ell)}} = \sigma_2^2 + O\left(K_{00}^{(\ell)}\right) = \sigma_2^2 + O\left(\frac{1}{\ell}\right), \tag{10.73}$$

remembering our notation $\sigma_2 \equiv \sigma''(0)$.

Next, we also have to make a choice about the training hyperparameters $\lambda_b^{(\ell)}$ and $\lambda_W^{(\ell)}$. Indeed, the depth scaling of the generalization error will depend on these hyperparameters, a fact that should not be surprising: we expect the selection of our relative learning rates to affect the performance of our model. Let's first follow the guidance of §9.4 where we discussed an *equivalence principle* for learning rates, and set these training hyperparameters according to (9.95), i.e., (9.70) multiplied by $L^{p_{\perp}-1}$:

$$\lambda_b^{(\ell)} = \widetilde{\lambda}_b \left(\frac{1}{\ell}\right)^{p_{\perp}} L^{p_{\perp}-1}, \qquad \lambda_W^{(\ell)} = \widetilde{\lambda}_W \left(\frac{L}{\ell}\right)^{p_{\perp}-1}. \tag{10.74}$$

With such a choice, we have an asymptotic solution for the midpoint frozen NTK, which is the same solution as in (9.71) up to a multiplication by $L^{p_{\perp}-1}$:

$$\Theta_{00}^{(\ell)} = \left[\widetilde{\lambda}_b + \frac{\widetilde{\lambda}_W \sigma_1^2}{(-a_1)}\right] \left(\frac{L}{\ell}\right)^{p_{\perp}-1} + \cdots . \tag{10.75}$$

Further plugging these results (10.71)–(10.75) into (10.70), we get

$$\begin{aligned} \delta\delta\Theta_{[2]}^{(\ell+1)} = &\left[1 - \frac{p_{\perp}}{\ell} + \cdots\right] \delta\delta\Theta_{[2]}^{(\ell)} \\ &+ \delta^2 \left\{ \widetilde{\lambda}_W \sigma_1^2 + \frac{\sigma_2^2}{\sigma_1^2} \left[\widetilde{\lambda}_b + \frac{\widetilde{\lambda}_W \sigma_1^2}{(-a_1)}\right] \right\} L^{p_{\perp}-1} \left(\frac{1}{\ell}\right)^{2p_{\perp}-1} + \cdots . \end{aligned} \tag{10.76}$$

With our usual methods, we can solve this recursion in the asymptotically large-ℓ limit with

$$\delta\delta\Theta_{[2]}^{(\ell)} = \delta^2 \frac{L^{p_{\perp}-1}}{(2-p_{\perp})} \left\{ \widetilde{\lambda}_W \sigma_1^2 + \frac{\sigma_2^2}{\sigma_1^2} \left[\widetilde{\lambda}_b + \frac{\widetilde{\lambda}_W \sigma_1^2}{(-a_1)}\right] \right\} \left(\frac{1}{\ell}\right)^{2p_{\perp}-2} + \cdots . \tag{10.77}$$

Finally, taking the ratio of the midpoint frozen NTK (10.75) and the perpendicular perturbation (10.77) and evaluating at the output layer $\ell = L$, we find the overall network depth dependence for our robustness measure:

$$\frac{-2\delta\delta\Theta_{[2]}^{(L)}}{\Theta_{00}^{(L)}} = \frac{2\delta^2}{(p_{\perp}-2)} \left\{ \frac{\widetilde{\lambda}_W \sigma_1^2}{\left[\widetilde{\lambda}_b + (\widetilde{\lambda}_W \sigma_1^2)/(-a_1)\right]} + \frac{\sigma_2^2}{\sigma_1^2} \right\} L^{1-p_{\perp}} \propto L^{1-p_{\perp}}. \tag{10.78}$$

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

This is astonishing: the desire to keep the robustness measure of order one for very deep networks exactly picks out activation functions in this universality class with $p_\perp = 1$![33] Such a condition is satisfied by any odd $K^\star = 0$ activation function, such as `tanh` and `sin`, both of which we've been discussing prominently throughout the book.

Now, let's zoom out for a moment and reflect on these calculations more broadly. Somewhat miraculously, the theoretically-motivated tuning of all our hyperparameters – *criticality* for the initialization hyperparameters and the learning rate *equivalence principle* for the training hyperparameters – has led to the most practically-optimal solution for the generalization error in this one-training-one-test setting, keeping the bias–variance tradeoff in check. Even more importantly, these choices and solutions are robust across many different network widths and depths, making them quite useful for experimentation and the scaling up of models.[34] Of course, there really was no miracle: our theoretical principles were practically motivated from the start.

Now that we understand what we *should* do, let's discuss a different choice of training hyperparameters that we *should not* make. Had we not followed the learning rate equivalence principle, perhaps we would have just made both weight and bias learning rates layer independent as $\lambda_b^{(\ell)} = \lambda_b$ and $\lambda_W^{(\ell)} = \lambda_W$. Let's see what happens then, specializing to odd activation functions with $\sigma_2 = 0$ and $p_\perp = 1$ for simplicity. In this case, our general formal solution for the single-input frozen NTK (9.69) reduces to

$$\Theta_{00}^{(\ell)} = \left(\frac{\lambda_b}{2}\right)\ell + \cdots , \tag{10.79}$$

with a linear dependence on the layer ℓ, while the same calculation as above with $\sigma_2 = 0$ and $p_\perp = 1$ in mind gives a layer-independent constant asymptotic solution for $\delta\delta\Theta_{[2]}^{(\ell)}$:

$$\delta\delta\Theta_{[2]}^{(\ell)} = \lambda_W \sigma_1^2 \delta^2 + \cdots . \tag{10.80}$$

Combined, our robustness measure becomes

$$\frac{-2\delta\delta\Theta_{[2]}^{(L)}}{\Theta_{00}^{(L)}} = \left[\frac{-4\lambda_W \sigma_1^2 \delta^2}{\lambda_b}\right]\frac{1}{L} + \cdots , \tag{10.81}$$

which is slowly but surely decaying with the overall depth L of the network. (The consideration of more general activation functions with $p_\perp \neq 1$ doesn't change this conclusion.) Therefore, this choice of the training hyperparameters is polynomially suboptimal compared to the choice based on our equivalence principle.[35]

[33] Since $p_\perp = b_1/a_1$, cf. (5.99); $b_1 \geq a_1$, cf. (5.86) and (5.88); and $a_1 < 0$, cf. (5.92), these overall imply that $p_\perp \leq 1$ for any $K^\star = 0$ activation function. In particular, for a non-odd activation function with $\sigma_2 \neq 0$, the exponent for perpendicular perturbations is strictly less than one, $p_\perp < 1$, and thus the bias term in the generalization error (10.56) will grow with network depth L.

[34] These tunings are more or less still valid even as we relax the infinite-width requirement to allow nonzero aspect ratio, $L/n \ll 1$.

[35] We leave it to the reader to see how disastrous things would be – in terms of our one-training-one-test generalization error – if we had decided not to rescale the weight learning rate by the widths of the previous layer as in (8.5), leading to an even more extreme violation of the learning rate equivalence principle.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Reflecting back, when we first discussed the learning rate equivalence principle by staring at our formal solution (9.69), we were motivated by the desire to ensure equal contributions to the NTK from each layer. Then in §9.4 we realized that such choices solve a polynomial version of the exploding and vanishing gradient problem. Here we see the downstream consequences of those choices through the lens of generalization error, giving a solid support for the equivalence principle according to our quantitative measure of training success.

Scale-Invariant Universality Class

To analyze scale-invariant activation functions, we need to use results from our finite-angle analysis in §5.5. In particular, the Gaussian expectation $\langle \sigma'' \sigma'' \rangle$ that appeared in the δ expansion of $\langle \sigma' \sigma' \rangle$ in (10.69) is singular for nonlinear scale-invariant functions due to the kink at the origin, and we promised we'd have to recall our finite-angle results when such a singularity occurs.

Keeping our promise to you, let's recall a bunch of things from that section. First, we decomposed the two-input kernel matrix as (5.146),

$$K^{(\ell)}_{\delta_1\delta_2} = \begin{pmatrix} K^{(\ell)}_{++} & K^{(\ell)}_{+-} \\ K^{(\ell)}_{-+} & K^{(\ell)}_{--} \end{pmatrix} = K^{(\ell)}_d \begin{pmatrix} 1 & \cos\left(\psi^{(\ell)}\right) \\ \cos\left(\psi^{(\ell)}\right) & 1 \end{pmatrix}, \qquad \psi^{(\ell)} \in [0, \pi], \tag{10.82}$$

with two dynamical variables being the diagonal kernel $K^{(\ell)}_d$ and the polar angle $\psi^{(\ell)}$. With this parametrization in mind, let us reprint a bunch of the previous results that we'll need, (5.60), (5.62), (5.159), and (5.161):

$$\langle \sigma_+ \sigma_+ \rangle_{K^{(\ell)}} = \langle \sigma_- \sigma_- \rangle_{K^{(\ell)}} = A_2 K^{(\ell)}_d, \tag{10.83}$$

$$C_W \langle \sigma'_+ \sigma'_+ \rangle_{K^{(\ell)}} = C_W \langle \sigma'_- \sigma'_- \rangle_{K^{(\ell)}} = C_W A_2 \equiv \chi, \tag{10.84}$$

$$\langle \sigma_+ \sigma_- \rangle_{K^{(\ell)}} = A_2 K^{(\ell)}_d \left\{ \cos\left(\psi^{(\ell)}\right) + \rho \left[\sin\left(\psi^{(\ell)}\right) - \psi^{(\ell)} \cos\left(\psi^{(\ell)}\right) \right] \right\}, \tag{10.85}$$

$$C_W \langle \sigma'_+ \sigma'_- \rangle_{K^{(\ell)}} = \chi (1 - \rho \psi^{(\ell)}), \tag{10.86}$$

where $A_2 \equiv (a_+^2 + a_-^2)/2$, $\rho \equiv \frac{1}{\pi} \frac{(a_+ - a_-)^2}{(a_+^2 + a_-^2)}$, and a_+ and a_- are the two constants that define the particular activation function (though by now you know that by heart).

Let us now make a similar decomposition for the frozen NTK as

$$\Theta^{(\ell)}_{\delta_1\delta_2} = \begin{pmatrix} \Theta^{(\ell)}_{++} & \Theta^{(\ell)}_{+-} \\ \Theta^{(\ell)}_{-+} & \Theta^{(\ell)}_{--} \end{pmatrix} = \Theta^{(\ell)}_d \begin{pmatrix} 1 & \cos\left(\zeta^{(\ell)}\right) \\ \cos\left(\zeta^{(\ell)}\right) & 1 \end{pmatrix}, \qquad \zeta^{(\ell)} \in [0, \pi], \tag{10.87}$$

with a diagonal frozen NTK $\Theta^{(\ell)}_d$ and another polar angle $\zeta^{(\ell)}$.

Then, plugging this decomposition (10.87) and recollected results (10.83)–(10.86) into the frozen NTK recursion (10.64), we get coupled recursions for the frozen NTK, cast in our finite-angle parameterization:

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

$$\Theta_d^{(\ell+1)} = \chi \Theta_d^{(\ell)} + \lambda_b^{(\ell+1)} + \lambda_W^{(\ell+1)} A_2 K_d^{(\ell)} , \tag{10.88}$$

$$\begin{aligned} \Theta_d^{(\ell+1)} \cos\left(\zeta^{(\ell+1)}\right) &= \chi(1 - \rho\psi^{(\ell)})\Theta_d^{(\ell)} \cos\left(\zeta^{(\ell)}\right) \\ &\quad + \lambda_b^{(\ell+1)} + \lambda_W^{(\ell+1)} A_2 K_d^{(\ell)} \left\{ \cos\left(\psi^{(\ell)}\right) + \rho \left[\sin\left(\psi^{(\ell)}\right) - \psi^{(\ell)} \cos\left(\psi^{(\ell)}\right) \right] \right\} . \end{aligned} \tag{10.89}$$

We see here in the off-diagonal recursion (10.89) a finite-angle analog of what we saw perturbatively for the $K^\star = 0$ universality class in the infinitesimal-angle recursion (10.70): the polar angle for the kernel $\psi^{(\ell)}$ sources the finite angle for the frozen NTK $\zeta^{(\ell)}$. Said another way, the exponential growth and decay of the kernel angle $\psi^{(\ell)}$ – at least for small enough angle – are linked to the exponential growth and decay of the frozen-NTK angle $\zeta^{(\ell)}$, which are in turn linked to the generalized bias–variance tradeoff.

With that chain of links in mind (as well as parallel discussions of similar issues in almost every other chapter of this book), it's natural that we should set our initialization hyperparameters by tuning to criticality: $\chi = 1$. With this choice, we recall the critical solutions from our finite-angle analysis of the kernel in §5.5:

$$K_d^{(\ell)} = K_d^\star , \qquad \psi^{(\ell)} = \left(\frac{3}{\rho}\right) \frac{1}{\ell} + \cdots , \tag{10.90}$$

where $K_d^\star$ is exactly constant, set by the first layer. Additionally, having already made the case for the learning rate equivalence principle when discussing the $K^\star = 0$ universality class, let's just simplify our discussion here by setting training hyperparameters according to that equivalence principle for scale-invariant activations (9.94): $\lambda_b^{(\ell)} = \widetilde{\lambda}_b / L$ and $\lambda_W^{(\ell)} = \widetilde{\lambda}_W / L$. With this choice, we see that

$$\Theta_d^{(\ell)} = \left(\widetilde{\lambda}_b + \widetilde{\lambda}_W A_2 K_d^\star \right) \frac{\ell}{L} \tag{10.91}$$

solves the recursion for the diagonal frozen NTK (10.88) with the initial condition (9.7). Importantly, here ℓ refers to a particular layer of the network, while L is the overall network depth.[36]

Plugging our choice of learning rates, our kernel solution (10.90), and NTK solution (10.91) into the finite-angle recursion (10.89), we get after a bit of rearranging

$$\cos\left(\zeta^{(\ell+1)}\right) = \left(1 - \frac{4}{\ell} + \cdots\right) \cos\left(\zeta^{(\ell)}\right) + \left(\frac{1}{\ell} + \cdots\right) , \tag{10.92}$$

which we can see easily is solved by an everything-independent constant

$$\cos\left(\zeta^{(\ell)}\right) = \frac{1}{4} + \cdots . \tag{10.93}$$

[36] The frozen NTK solution (10.91) is identical to our previous single-input solution (9.44); here we have just rescaled the bias and weight learning rates by the overall depth, $\lambda_b = \widetilde{\lambda}_b / L$ and $\lambda_W = \widetilde{\lambda}_W / L$, as required by the equivalence principle (9.94).

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Thus, our robustness measure (10.57) in the bias term of the generalization error for nonlinear scale-invariant activation functions is given by a simple order-one number:

$$\frac{\Theta_{-+}^{(L)}}{\Theta_{++}^{(L)}} - 1 = \cos\left(\zeta^{(L)}\right) - 1 = -\frac{3}{4} + \cdots . \tag{10.94}$$

Similarly, given that the nearby-input analysis can break down for nonlinear scale-invariant activations, let's use our finite-angle analysis here to also work out the variance term of the generalization error (10.63); plugging in the asymptotic falloff for the kernel (5.167) and (5.168) as well as using (10.94) for the frozen NTK, we get

$$\begin{aligned} \text{Cov}\left[z_{i;-}^{(L)}(T),\, z_{i;-}^{(L)}(T)\right] &= K_{--} - 2\frac{\Theta_{-+}}{\Theta_{++}}K_{-+} + \left(\frac{\Theta_{-+}}{\Theta_{++}}\right)^2 K_{++} \\ &= K_d^\star \left[1 - \cos\left(\zeta^{(L)}\right)\right]^2 = \frac{9}{16}K_d^\star + \cdots . \end{aligned} \tag{10.95}$$

Unlike the previous case for $K^\star = 0$ activations, these asymptotic results for the generalization error, (10.94) and (10.95), don't depend on the training hyperparameters $\widetilde{\lambda}_b$ and $\widetilde{\lambda}_W$, nor do they depend on a constant like δ^2 that knows about the separation of the test and training points. (However, just as we discussed for $\psi^{(\ell)}$ in §5.5, the depth at which these asymptotic results become valid does depend on δ^2, the input norm, the activation function, and the training hyperparameters.) Nonetheless, again with the correct tuning of our hyperparameters based on the principles of criticality and equivalence, we found a constant bias and variance, giving us the best possible tradeoff when training deep networks with nonlinear scale-invariant activations.[37]

Let us end with a special remark on deep linear networks. For these networks, we use the `linear` activation function with $a_+ = a_-$ and hence have $\rho = 0$. In particular, we saw in §5.5 that not only was the diagonal kernel preserved at criticality, but the polar angle was preserved as well: $K_d^{(\ell)} = K_d^\star$ and $\psi^{(\ell)} = \psi^\star$. Noting this, the off-diagonal recursion for the frozen NTK (10.89) then becomes

$$\Theta_d^{(\ell+1)} \cos\left(\zeta^{(\ell+1)}\right) = \Theta_d^{(\ell)} \cos\left(\zeta^{(\ell)}\right) + \frac{\widetilde{\lambda}_b}{L} + \frac{\widetilde{\lambda}_W}{L} A_2 K_d^\star \cos(\psi^\star) . \tag{10.96}$$

This recursion is exactly solved by

$$\Theta_d^{(\ell)} \cos\left(\zeta^{(\ell)}\right) = \left[\widetilde{\lambda}_b + \widetilde{\lambda}_W A_2 K_d^\star \cos(\psi^\star)\right] \frac{(\ell-1)}{L} + \Theta_d^{(1)} \cos\left(\zeta^{(1)}\right) . \tag{10.97}$$

Dividing this result by our solution for the diagonal frozen NTK (10.91) then gives

$$\cos\left(\zeta^{(\ell)}\right) = \frac{\widetilde{\lambda}_b + \widetilde{\lambda}_W A_2 K_d^\star \cos(\psi^\star)}{\widetilde{\lambda}_b + \widetilde{\lambda}_W A_2 K_d^\star} + \cdots , \tag{10.98}$$

[37] It is worth noting what happens with the special case of exact Bayesian inference where the only nonzero learning rates are in the last layer in order to set $\Theta^{(L)} = K^{(L)}$. In that case, the robustness measure is given by $\cos\left(\psi^{(L)}\right) - 1 = O\left(1/\ell^2\right)$. Given this decay with depth, we see that the restricted Bayesian case is clearly inferior to an ensemble of networks that are fully trained via gradient descent with uniform learning rates across layers.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

which polynomially asymptotes to a constant. Unlike the case for the nonlinear scale-invariant activation functions, this constant depends on the observables in the first layer in a rather detailed manner, naturally connecting the generalization properties of the network to the input. The real limitation of the deep linear networks becomes immediately apparent upon considering their (in)ability to interpolate/extrapolate, which we'll analyze next for linearly and nonlinearly activated MLPs.

10.3.2 Interpolation and Extrapolation

Rather than focusing primarily on our evaluation criteria for successful training, the generalization error, in this subsection we will focus more on the kinds of functions that our trained neural networks actually compute. This analysis will enable us to consider the *inductive bias* of different activation functions and tell us how to relate the properties of those activation functions to the properties of the dataset and function that we're trying to approximate.

In the previous subsection we asked: given the true output $y_{i;+}$ for an input $x_{i;+}$, what does a fully-trained MLP in the infinite-width limit predict for the output of a nearby input $x_{i;-}$? Here, we up the ante and ask: given the true outputs $y_{i;\pm}$ for *two* inputs $x_{i;\pm} = x_{i;0} \pm \frac{\delta x_i}{2}$, what is the prediction for a one-parameter family of test inputs,

$$s x_{i;+} + (1-s) x_{i;-} = x_{i;0} + \frac{(2s-1)}{2} \delta x_i \equiv x_{i;(2s-1)}, \tag{10.99}$$

that sit on a line passing through $x_{i;+}$ and $x_{i;-}$? When our parameter s is inside the unit interval $s \in [0, 1]$, this is a question about neural-network **interpolation**; for s outside the unit interval, it's a question about **extrapolation**. For general s, let's refer to this collectively as $*$**-polation**.

First we'll perform a little exercise in $*$-polation with deep linear networks to see what networks with `linear` activation functions do. Accordingly, we'll see concretely how deep linear networks approximate a very limited set of functions. Then we'll follow up by assessing smooth nonlinear networks.

Linear $*$-Polation by Deep Linear Networks

There's a very simple way to see how $*$-polation works for deep linear networks. If we recall for a moment (and for one last time) the forward equation for deep linear networks (3.1),

$$z_{i;\alpha}^{(\ell+1)} = b_i^{(\ell+1)} + \sum_{j=1}^{n_\ell} W_{ij}^{(\ell+1)} z_{j;\alpha}^{(\ell)}, \tag{10.100}$$

with $z_{j;\alpha}^{(0)} \equiv x_{j;\alpha}$, it's clear that the linear structure in the input (10.99) will be preserved from layer to layer. That is, given an ℓ-th-layer preactivation of the form

$$z_{i;(2s-1)}^{(\ell)} = s z_{i;+}^{(\ell)} + (1-s) z_{i;-}^{(\ell)} \tag{10.101}$$

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

that has such a linear structure, we then have for the next layer

$$
\begin{aligned}
z^{(\ell+1)}_{i;(2s-1)} &= b^{(\ell+1)}_i + \sum_{j=1}^{n_\ell} W^{(\ell+1)}_{ij} \left[s z^{(\ell)}_{j;+} + (1-s) z^{(\ell)}_{j;-} \right] \qquad (10.102)\\
&= s \left(b^{(\ell+1)}_i + \sum_{j=1}^{n_\ell} W^{(\ell+1)}_{ij} z^{(\ell)}_{j;+} \right) + (1-s) \left(b^{(\ell+1)}_i + \sum_{j=1}^{n_\ell} W^{(\ell+1)}_{ij} z^{(\ell)}_{j;-} \right) \\
&= s z^{(\ell+1)}_{i;+} + (1-s) z^{(\ell+1)}_{i;-} \,,
\end{aligned}
$$

which still respects the linear structure. This is just a direct consequence of the fact that deep linear networks compute linear functions of their input.

Therefore, for a test input that's a linear sum of our two training points (10.99), the network will output the linear sum of the network outputs on the two individual training points:

$$
z^{(L)}_{i;(2s-1)} = s z^{(L)}_{i;+} + (1-s) z^{(L)}_{i;-}. \qquad (10.103)
$$

This equation holds at initialization as well as at the end of training, which means that any particular fully-trained deep linear network will $*$-polate as

$$
z^{(L)}_{i;(2s-1)}(T) = s y_{i;+} + (1-s) y_{i;-}, \qquad (10.104)
$$

since the fully-trained network output will equal the true output for any element in the training set: $z^{(L)}_{i;\pm}(T) = y_{i;\pm}$. With this, we see that fully-trained deep linear networks *linearly* $*$-polate, no matter what. This is both intuitive and pretty obvious: as deep linear networks perform linear transformations, they can only compute linear functions.

Of course, this is exactly what we said when we studied deep linear networks way back in §3. Here, we explicitly see *why* these networks are limited after training, by showing the (limited) way in which they can use training examples to make predictions. Accordingly, if the function you're trying to approximate is a linear function of the input data, then deep linear networks are a great modeling choice. If the function is nonlinear, we'll have to consider nonlinear activation functions. It's not that deep.

Nonlinear $*$-Polation by Smooth Nonlinear Deep Networks

We'll have to work a little harder to see what nonlinear networks do.[38] In the last section, we saw that the output of a fully-trained network is given by the stochastic kernel prediction equation (10.39), which we reprint here for convenience:

$$
z^{(L)}_{i;\dot\beta}(T) = z^{(L)}_{i;\dot\beta} - \sum_{\tilde\alpha_1,\tilde\alpha_2 \in \mathcal{A}} \Theta^{(L)}_{\dot\beta\tilde\alpha_1} \widetilde{\Theta}^{\tilde\alpha_1\tilde\alpha_2}_{(L)} \left(z^{(L)}_{i;\tilde\alpha_2} - y_{i;\tilde\alpha_2} \right). \qquad (10.105)
$$

[38] We would have to work even harder to see what nonlinear scale-invariant activation functions do, so here we'll focus on smooth nonlinear activation functions. For such nonlinear scale-invariant activation functions with kinks, since the $*$-polated input $x_{i;(2s-1)}$ does not have the same norm as $x_{i;\pm}$ for $s \neq 0, 1$, we would need to extend the finite-angle analysis from §5.5 to the case of unequal input norms. This is left as a challenge in pedagogy to future deep-learning book authors.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Thus, we see that to study $*$-polation more generally, we will need to evaluate elements of the frozen NTK between our test and training set, $\Theta^{(L)}_{(2s-1)\pm}$, and also need to invert the two-by-two submatrix of the frozen NTK on the training set only, $\widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2}_{(L)}$.

This latter inversion can be easily completed with the standard textbook formula for the inverse of a two-by-two matrix,

$$\widetilde{\Theta}^{\tilde{\alpha}_1\tilde{\alpha}_2} = \frac{1}{\Theta_{++}\Theta_{--} - \Theta_{+-}^2} \begin{pmatrix} \Theta_{--} & -\Theta_{+-} \\ -\Theta_{+-} & \Theta_{++} \end{pmatrix}, \tag{10.106}$$

where here we've also used the symmetry $\Theta_{+-} = \Theta_{-+}$ and further dropped the layer indices. For the rest of this section, we will always assume that these frozen NTKs are evaluated at the output layer.

Next, to compute the off-diagonal elements between the training set and the test set $\Theta_{(2s-1)\pm}$, we'll need to generalize our δ expansion a bit. Let's first recall our expressions for the components of the frozen NTK in $\gamma^{[a]}$ basis, (10.58), and then plug in the δ expansion we performed on the two-by-two submatrix $\widetilde{\Theta}_{\tilde{\alpha}_1\tilde{\alpha}_2}$, (10.59)–(10.61), which gives

$$\Theta_{\pm\pm} = \Theta_{00} \pm \delta\Theta_{[1]} + \Big(\delta\delta\Theta_{[2]} + \delta\delta\Theta_{[0]}\Big) + O\Big(\delta^3\Big) \ , \tag{10.107}$$

$$\Theta_{\pm\mp} = \Theta_{00} + \Big(-\delta\delta\Theta_{[2]} + \delta\delta\Theta_{[0]}\Big) + O\Big(\delta^3\Big) \tag{10.108}$$

for the pair of inputs $x_{i;\pm} = x_{i;0} \pm \frac{\delta x_i}{2}$. Let's now consider a pair of perturbed inputs of a more general form

$$x_{i;\epsilon_1} \equiv x_{i;0} + \frac{\epsilon_1}{2}\delta x_i \,, \qquad x_{i;\epsilon_2} \equiv x_{i;0} + \frac{\epsilon_2}{2}\delta x_i. \tag{10.109}$$

Note that picking $\epsilon_{1,2}$ from ± 1 reduces them to $x_{i;\pm}$, while the new case of interest, $\Theta_{(2s-1)\pm}$, corresponds to setting $\epsilon_1 = (2s-1)$ and $\epsilon_2 = \pm 1$. For a generic pair of inputs (10.109), the δ expansion gets modified as

$$\begin{aligned}\Theta_{\epsilon_1\epsilon_2} = \Theta_{00} &+ \left(\frac{\epsilon_1+\epsilon_2}{2}\right)\delta\Theta_{[1]} + \left(\frac{\epsilon_1+\epsilon_2}{2}\right)^2\Big(\delta\delta\Theta_{[2]} + \delta\delta\Theta_{[0]}\Big) \\ &+ \left(\frac{\epsilon_1-\epsilon_2}{2}\right)^2\Big(-\delta\delta\Theta_{[2]} + \delta\delta\Theta_{[0]}\Big) + O\Big(\epsilon^3\delta^3\Big) \ .\end{aligned} \tag{10.110}$$

To see why this is the correct expression, note that *(i)* each term has the right scaling with $\epsilon_{1,2}$; *(ii)* for $\epsilon_1 = \epsilon_2 = \pm 1$, we correctly recover the expression for $\Theta_{\epsilon_1\epsilon_2} = \Theta_{\pm\pm}$ (10.107); *(iii)* for $\epsilon_1 = -\epsilon_2 = \pm 1$, we correctly recover the expression for $\Theta_{\epsilon_1\epsilon_2} = \Theta_{\pm\mp}$ (10.108); and *(iv)* the expression is symmetric under $\epsilon_1 \leftrightarrow \epsilon_2$. The frozen NTK component $\Theta_{\epsilon_1\epsilon_2}$ must satisfy these four constraints, and the expression (10.110) is the unique formula that satisfies them all.

Applying this formula to evaluate $\Theta_{(2s-1)\pm}$ and simplifying a bit, we find

$$\Theta_{(2s-1)\pm} = s\Theta_{\pm+} + (1-s)\Theta_{\pm-} - 2s(1-s)\delta\delta\Theta_{[0]} + O\Big(\delta^3\Big) . \tag{10.111}$$

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

As we'll see, the key to nonlinear $*$-polation, at least for nearby inputs, is in the $\delta\delta\Theta_{[0]}$ term.[39] Firstly, it's clear that this term nonlinearly depends on the test input, as evidenced by the $s(1-s)$ prefactor. Indeed, you can go back and check that this term identically vanishes for deep linear networks, i.e., for those networks we simply have $\Theta_{(2s-1)\pm} = s\Theta_{\pm+} + (1-s)\Theta_{\pm-}$.[40] With that in mind, it also helps to decompose the initial preactivation into linear and nonlinear pieces as

$$z^{(L)}_{i;(2s-1)} = sz^{(L)}_{i;+} + (1-s)z^{(L)}_{i;-} + \left[z^{(L)}_{i;(2s-1)} - sz^{(L)}_{i;+} - (1-s)z^{(L)}_{i;-}\right]. \tag{10.112}$$

Here, the second term vanishes for deep linear networks, as per (10.103), and so in general it captures nonlinearity of the network output at initialization.

Plugging (10.106), (10.111), and (10.112) into our kernel prediction formula (10.105), we see that our fully-trained prediction on the test input $x_{i;(2s-1)} = sx_{i;+} + (1-s)x_{i;-}$ is given by

$$\begin{aligned} &z^{(L)}_{i;(2s-1)}(T) \\ &\quad= \left[z^{(L)}_{i;(2s-1)} - sz^{(L)}_{i;+} - (1-s)z^{(L)}_{i;-}\right] + \left[sy_{i;+} + (1-s)y_{i;-}\right] \\ &\qquad - s(1-s)\left[\frac{2\delta\delta\Theta_{[0]}}{\Theta_{00}\delta\delta\Theta_{[2]} - \delta\Theta^2_{[1]}}\right]\Big[2\delta\delta\Theta_{[2]}\left(z^{(L)}_{i;+} + z^{(L)}_{i;-} + y_{i;+} + y_{i;-}\right) \\ &\qquad\qquad - \delta\Theta_{[1]}\left(z^{(L)}_{i;+} - z^{(L)}_{i;-} + y_{i;+} - y_{i;-}\right)\Big] + O\left(\delta^3\right). \end{aligned} \tag{10.113}$$

Comparing with our linear $*$-polation formula (10.104), we see that both the first and last terms are new: nonlinear networks can *nonlinearly* $*$-polate! Interestingly, the fully-trained $*$-polation for nonlinear activation functions depends on the network output at initialization through the nonlinearity $z^{(L)}_{i;(2s-1)} - sz^{(L)}_{i;+} - (1-s)z^{(L)}_{i;-}$; in contrast, for deep linear networks the $*$-polation only depended on the true output of the training examples.

As a particular illustration of this formula, consider the case when the two training inputs have the same norm. In such a case, $\Theta_{[1]} = 0$, and we find a much simpler formula:

$$\begin{aligned} z^{(L)}_{i;(2s-1)}(T) = &\left[z^{(L)}_{i;(2s-1)} - sz^{(L)}_{i;+} - (1-s)z^{(L)}_{i;-}\right] + \left[sy_{i;+} + (1-s)y_{i;-}\right] \\ &- 4s(1-s)\left(\frac{\delta\delta\Theta_{[0]}}{\Theta_{00}}\right)\left(z^{(L)}_{i;+} + z^{(L)}_{i;-} + y_{i;+} + y_{i;-}\right) + O\left(\delta^3\right). \end{aligned} \tag{10.114}$$

[39] N.B. $\delta\delta\Theta_{[0]}$ is very different from $\delta\delta\Theta_{[2]}$: the former is the second term in the expansion of the $\gamma^{[0]}$ component $\Theta_{[0]}$, cf. (10.59), while the latter is the first term in the expansion of the $\gamma^{[2]}$ component $\Theta_{[2]}$, cf. (10.61).

[40] To see this quickly, note that both the first-layer metric (4.8) and the first-layer NTK (8.23) are bilinear in the two inputs and that such bilinear structure is preserved under the recursions for deep linear networks: $K^{(\ell+1)}_{\delta_1\delta_2} = C_b^{(\ell+1)} + C_W^{(\ell+1)}K^{(\ell)}_{\delta_1\delta_2}$, cf. (4.118), and $\Theta^{(\ell+1)}_{\delta_1\delta_2} = \lambda_b^{(\ell+1)} + \lambda_W^{(\ell+1)}K^{(\ell)}_{\delta_1\delta_2} + C_W^{(\ell+1)}\Theta^{(\ell)}_{\delta_1\delta_2}$, cf. (9.5).

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

Averaging over our ensemble, this prediction has a mean

$$m^{\infty}_{i;(2s-1)} = sy_{i;+} + (1-s)y_{i;-} - 4s(1-s)\left(\frac{\delta\delta\Theta_{[0]}}{\Theta_{00}}\right)(y_{i;+}+y_{i;-}) + O\left(\delta^3\right). \quad (10.115)$$

Here the first term in (10.114) that captured the nonlinearity of the network output at initialization vanished under the expectation, and so the nonlinearity of the $*$-polation mean is entirely captured by the dimensionless ratio $\delta\delta\Theta_{[0]}/\Theta_{00}$.

So, what kind of a function is our fully-trained infinite-width nonlinear neural network computing? To assess this, note that the ratio $\delta\delta\Theta_{[0]}/\Theta_{00}$ captures the *curvature* of the $*$-polation in the neighborhood of the training points.[41] This curvature encodes a nonuniversal *inductive bias* of the activation function and architecture indicating how this class of function approximators will generalize to novel data.

For a given task and dataset, some activation functions might produce a more desired type of $*$-polation. This can be measured directly via the bias term in the generalization error. Substituting in our equal norm expression for the mean (10.115),

$$\begin{aligned} m^{\infty}_{i;(2s-1)} - y_{i;(2s-1)} = &\left[y_{i;+} + (1-s)y_{i;-} - y_{i;(2s-1)}\right] \\ &- 4s(1-s)\left(\frac{\delta\delta\Theta_{[0]}}{\Theta_{00}}\right)(y_{i;+}+y_{i;-}) + O\left(\delta^3\right), \end{aligned} \quad (10.116)$$

we see that this generalization error bias decomposes into a comparison between the nonlinearity in the true output – given by the terms in the square brackets – and the network curvature around the midpoint of the true output $(y_{i;+}+y_{i;-})/2$. With this framing, deep linear networks promote a very particular type of inductive bias: only linear functions are computed. More generally, we could (but won't here) compute and solve a recursion for $\delta\delta\Theta_{[0]}$ for any particular activation function in order to learn more about the kinds of functions computed by deep networks with that activation function.

Finally, note that this analysis doesn't make any particular distinction between *interpolation* and *extrapolation* and also that as $s \to 0, 1$, the $*$-polation (10.113) reduces to $y_{i;\pm}$ with absolute certainty. In fact, in the neighborhood of $s = 0, 1$, the $*$-polation bias (10.116) has much in common with the prediction bias (10.56) and (10.62) that we saw in §10.3.1 when considering a training set consisting of only one training sample. Importantly, it is the most nearby training point that contributes the most to a test point's prediction.

Taken as a guide to thinking about larger training sets, the *local* nature of these predictions is highly suggestive of some ways to make further progress. On the one hand, we might be able to make theoretical progress on more complicated prediction formulae

[41] Note that as the two training samples begin to coincide, $x_{\pm} \to x_0$, the curvature vanishes quadratically, $\delta\delta\Theta_{[0]}/\Theta_{00} = O\left(\delta^2\right)$, and the closer the $*$-polation will be to a linear $*$-polation. Further applying our generalized δ expansion (10.110) to the kernel, we can show that the variance of the $*$-polation vanishes even more quickly in this coincident limit as

$$\mathbb{E}\left[z^{(L)}_{i;2s-1}(T)z^{(L)}_{i;2s-1}(T)\right] - \left(\mathbb{E}\left[z^{(L)}_{i;2s-1}(T)\right]\right)^2 = O\left(\delta^3\right). \quad (10.117)$$

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

by weighting the predictions given by nearby training points to a given test point, perhaps using an approximation from §10.3.1 when there's only one nearby training point and using $*$-polation (10.113) when there's a nearby pair. On the other hand, we might be able to make practical progress on training-set design – given the network's inductive biases – by using this kind of analysis to inform how best to sample training inputs over the data manifold.

10.4 Linear Models and Kernel Methods

Before we back off the infinite-width limit, let's take a section to place what we've done in this chapter into the broader context of machine learning. In the next chapter, such a context will help us understand the ways in which deep learning at finite width is qualitatively quite different from its infinite-width counterpart.

In particular, in this section we'll explain a *dual* way of thinking about the class of models that can be described by a kernel prediction formula such as (10.39). On the one hand, kernel predictions can be thought of as being made by $*$-polating the training data using the kernel. On the other hand, we can think of them as the output of a trained model that's linear in its parameters. The former perspective has been more natural to us, given that we always consider an ensemble over the model parameters and then integrate them out. So let's begin by explaining the latter *linear model* perspective.[42]

10.4.1 Linear Models

The simplest linear model – and perhaps the simplest machine learning model – is just a one-layer (i.e., zero-hidden-layer) network:

$$z_i(x_\delta; \theta) = b_i + \sum_{j=1}^{n_0} W_{ij} x_{j;\delta}. \tag{10.118}$$

While this model is linear in both the parameters $\theta = \{b_i, W_{ij}\}$ and the input $x_{j;\delta}$, the *linear* in *linear model* takes its name from the dependence on the parameters θ and not the input x. In particular, while the components of the input samples $x_{j;\delta}$ sometimes can serve as a reasonable set of features for function approximation, in general they do not. Indeed, considering how much ink we've already spilled on representation group flow and representation learning in the context of deep learning, it's natural to expect that we would need to (pre-)process the input data before it's useful for any machine learning task.

One traditional way to fix this, inherited from statistics, is to engineer better features. Such an approach was necessary when computers were less powerful and models had to be much simpler to optimize. For instance, in addition to the features x_j, perhaps it

[42] The connection between infinite-width networks trained by gradient descent and kernel methods was pointed out in [57] in the context of introducing the NTK. Following that, an extended discussion of such networks as linear models was given in [62].

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

would also be useful for the model to take into account features $x_j x_k$ that let us consider the dependence of one component upon another. More generally, we might design a fixed set of **feature functions** $\phi_j(x)$ that's meant to work well for the dataset $\mathcal{D}$ and the underlying task at hand.[43]

In this traditional approach, the hope is that all the complicated modeling work goes into the construction of these feature functions $\phi_j(x)$, and, if we do a good enough job, then its associated **linear model**,

$$z_i(x_\delta;\theta) = b_i + \sum_{j=1}^{n_f} W_{ij}\phi_j(x_\delta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta), \tag{10.119}$$

is simple to train, easy to interpret, and performs well on the desired task. Here, we've followed a customary notational reductionism, subsuming the bias vector into the weight matrix by setting $\phi_0(x) \equiv 1$ and $W_{i0} \equiv b_i$. Thus, the output $z_i(x;\theta)$ of a linear model depends linearly on the model parameters θ, consisting of a combined weight matrix W_{ij} of dimension $n_{\text{out}} \times (n_f + 1)$. We can still think of this model as a one-layer neural network, but in this case we pre-process each input with the function $\phi_j(x)$ before passing it through the network.

Now let's explain how to learn the optimal values for weight matrix $W^\star_{ij}$ given a training set $\mathcal{A}$. The most common approach is to minimize the MSE loss

$$\mathcal{L}_{\mathcal{A}}(\theta) = \frac{1}{2}\sum_{\tilde{\alpha}\in\mathcal{A}}\sum_{i=1}^{n_{\text{out}}}\left[y_{i;\tilde{\alpha}} - z_i(x_{\tilde{\alpha}};\theta)\right]^2 = \frac{1}{2}\sum_{\tilde{\alpha}\in\mathcal{A}}\sum_{i=1}^{n_{\text{out}}}\left[y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij}\phi_j(x_{\tilde{\alpha}})\right]^2. \tag{10.120}$$

Supervised learning with a linear model is known as **linear regression**, and – as the MSE loss of a linear model is necessarily quadratic in the model parameters – this is another case of an analytically-solvable learning problem (7.7). Taking the derivative of $\mathcal{L}_{\mathcal{A}}$ with respect to the parameters and setting it to zero, we get an implicit equation that determines the optimal weight matrix $W^\star_{ij}$:

$$\sum_{k=0}^{n_f} W^\star_{ik}\left[\sum_{\tilde{\alpha}\in\mathcal{A}}\phi_k(x_{\tilde{\alpha}})\phi_j(x_{\tilde{\alpha}})\right] = \sum_{\tilde{\alpha}\in\mathcal{A}} y_{i;\tilde{\alpha}}\phi_j(x_{\tilde{\alpha}}). \tag{10.121}$$

To solve this equation, let's define a symmetric (n_f+1)-by-(n_f+1) matrix of features,

$$M_{ij} \equiv \sum_{\tilde{\alpha}\in\mathcal{A}}\phi_i(x_{\tilde{\alpha}})\phi_j(x_{\tilde{\alpha}}), \tag{10.122}$$

with elements that give a pairwise aggregation of feature functions summed over all the training samples $\tilde{\alpha} \in \mathcal{A}$. Then, applying its inverse to both sides of the implicit expression (10.121), we find a solution:

[43] These types of feature functions are also useful if the input x is something abstract – such as a document of text – and thus needs to be transformed into a numerical vector before it can be processed by a parameterized model.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

$$W^\star_{ij} = \sum_{k=0}^{n_f} \sum_{\tilde{\alpha}\in\mathcal{A}} y_{i;\tilde{\alpha}} \phi_k(x_{\tilde{\alpha}}) \left(M^{-1}\right)_{kj} . \tag{10.123}$$

Notice that the solution depends on the training set, linearly for the true function values $y_{i;\tilde{\alpha}}$ and in a more complicated way on the input features $\phi_k(x_{\tilde{\alpha}})$.[44] Finally, we can use this fully-trained linear model with its associated optimal parameters $W^\star_{ij}$ to make predictions on novel test-set inputs $x_{\dot{\beta}}$ as

$$z_i(x_{\dot{\beta}}; \theta^\star) = \sum_{j=0}^{n_f} W^\star_{ij} \phi_j(x_{\dot{\beta}}) , \tag{10.124}$$

giving us a closed-form solution for our linear regression problem. Importantly, after learning is complete we can simply store the optimal parameters $W^\star_{ij}$ and forget about the training data.

10.4.2 Kernel Methods

While this is all very easy, it's less familiar in our book since we typically do not work explicitly with the parameters. To cast our linear model into a more familiar form, let's consider a *dual* expression for the solution. First, let's substitute our expression for the optimal parameters $W^\star_{ij}$, (10.123), into our linear regression solution, (10.124), giving

$$z_i(x_{\dot{\beta}}; \theta^\star) = \sum_{\tilde{\alpha}\in\mathcal{A}} \left[\sum_{j,k=0}^{n_f} \phi_j(x_{\dot{\beta}}) \left(M^{-1}\right)_{jk} \phi_k(x_{\tilde{\alpha}}) \right] y_{i;\tilde{\alpha}}. \tag{10.125}$$

Note that the expression in the square brackets involves the inversion of an $(n_f + 1) \times (n_f + 1)$-dimensional matrix M_{ij}, which was required to obtain the optimal parameters $W^\star_{ij}$. This works well if the number of features is small, but if the number of feature functions we defined is very large, $n_f \gg 1$, then representing and inverting such a matrix might be computationally difficult.

However, it turns out that we actually don't need to do any of that. To see why, let us introduce a new $N_\mathcal{D} \times N_\mathcal{D}$-dimensional symmetric matrix:

$$k_{\delta_1\delta_2} \equiv k(x_{\delta_1}, x_{\delta_2}) \equiv \sum_{i=0}^{n_f} \phi_i(x_{\delta_1})\, \phi_i(x_{\delta_2}) . \tag{10.126}$$

[44] If the number of features $(n_f + 1)$ is larger than the size of the training set $N_\mathcal{A}$, then the model is *overparameterized*, and M_{ij} is not uniquely invertible. One scheme to specify the solution is to add a regularization term of the form $a \sum_{ij} W^2_{ij}$ to the loss (10.120), cf. footnote 21 in §10.2.4 for a related discussion of regularization for infinite-width networks. In this modified regression problem, we can then invert the regularized matrix

$$M_{ij} = 2a\, \delta_{ij} + \sum_{\tilde{\alpha}\in\mathcal{A}} \phi_i(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \tag{10.127}$$

and send the regulator to zero, $a \to 0^+$, at the end of our calculations. Note that either when the regulator a is kept finite or when we're in the *underparameterized* regime with $(n_f + 1) < N_\mathcal{A}$, the linear model will no longer reach zero training loss even when fully optimized.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

As an inner product of feature functions, $k_{\delta_1\delta_2}$ is a measure of similarity between two inputs $x_{i;\delta_1}$ and $x_{i;\delta_2}$ in feature space. Such a measure of similarity is called a **kernel**.[45] In a way that should feel very familiar, we'll also denote an $N_{\mathcal{A}}$-by-$N_{\mathcal{A}}$-dimensional submatrix of the kernel evaluated on the training set as $\widetilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_2}$ with a tilde. This lets us write its inverse as $\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2}$, which satisfies

$$\sum_{\tilde{\alpha}_2\in\mathcal{A}} \widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2}\widetilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_3} = \delta^{\tilde{\alpha}_1}_{\tilde{\alpha}_3}. \tag{10.128}$$

Note that given the definition of the kernel (10.126), for this inverse to exist and for this equation to hold, we must be in the *overparameterized* regime with $(n_f+1) \geq N_{\mathcal{A}}$.

Now with this, let's see how we might rearrange the factor in the square brackets of our solution (10.125). Multiplying it by the submatrix $\widetilde{k}_{\tilde{\alpha}\tilde{\alpha}_1}$, we can simplify this factor as

$$\begin{aligned}
&\sum_{\tilde{\alpha}\in\mathcal{A}}\left[\sum_{j,k=0}^{n_f}\phi_j(x_{\dot{\beta}})\left(M^{-1}\right)_{jk}\phi_k(x_{\tilde{\alpha}})\right]\widetilde{k}_{\tilde{\alpha}\tilde{\alpha}_1} \qquad (10.129)\\
&\quad=\sum_{\tilde{\alpha}\in\mathcal{A}}\sum_{j,k=0}^{n_f}\phi_j(x_{\dot{\beta}})\left(M^{-1}\right)_{jk}\phi_k(x_{\tilde{\alpha}})\sum_{i=0}^{n_f}\phi_i(x_{\tilde{\alpha}})\,\phi_i(x_{\tilde{\alpha}_1})\\
&\quad=\sum_{i,j,k=0}^{n_f}\phi_j(x_{\dot{\beta}})\left(M^{-1}\right)_{jk}M_{ki}\,\phi_i(x_{\tilde{\alpha}_1})\\
&\quad=\sum_{i=0}^{n_f}\phi_i(x_{\dot{\beta}})\phi_i(x_{\tilde{\alpha}_1})=k_{\dot{\beta}\tilde{\alpha}_1}.
\end{aligned}$$

To get this result, in the second line we plugged in the definition of the kernel (10.126), in the third line we performed the sum over $\tilde{\alpha}$ using the definition of the feature matrix M_{ij} (10.122), and in the last equality of the fourth line we again used the definition of the kernel. Finally, multiplying the first and last expressions by the inverse submatrix $\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2}$, we get a new representation for the factor in the square brackets,

$$\left[\sum_{j,k=0}^{n_f}\phi_j(x_{\dot{\beta}})\left(M^{-1}\right)_{jk}\phi_k(x_{\tilde{\alpha}_2})\right]=\sum_{\tilde{\alpha}_1\in\mathcal{A}}k_{\dot{\beta}\tilde{\alpha}_1}\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2}, \tag{10.130}$$

which lets us rewrite the prediction of our linear model (10.124) as

$$z_i(x_{\dot{\beta}};\theta^\star)=\sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}}k_{\dot{\beta}\tilde{\alpha}_1}\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2}y_{i;\tilde{\alpha}_2}. \tag{10.131}$$

[45] For instance, in the case of the simplest linear model (10.118), the kernel is just given by the inner product between the two inputs

$$k_{\delta_1\delta_2}\equiv\sum_{i=1}^{n_0}x_{i;\delta_1}x_{i;\delta_2}, \tag{10.132}$$

which is often called the *linear kernel*.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

When the prediction of a linear model is computed in this way, it's known as a *kernel machine* or a **kernel method**.

Note that in this dual expression of the solution, the optimal parameters $W^\star_{ij}$ and the feature functions $\phi_i(x)$ don't appear. Thus, we've successfully exchanged our feature-space quantities – an (n_f+1)-dimensional feature vector and the inverse of an $(n_f+1)\times(n_f+1)$-dimensional matrix – for sample-space quantities – an $N_{\mathcal{A}}$-dimensional vector $k_{\dot\beta\tilde\alpha_1}$ and the inverse of an $N_{\mathcal{A}}\times N_{\mathcal{A}}$-dimensional matrix $\widetilde{k}_{\tilde\alpha_1\tilde\alpha_2}$.[46] This works because in our solution (10.131), we actually only care about the inner product of the feature functions – i.e., the kernel – and not the values of the features themselves.

By writing the linear model's prediction in terms of the kernel in (10.131), we can interpret the prediction in terms of direct comparison with previously-seen examples. In particular, this solution computes the similarity of a new test input $x_{\dot\beta}$ with all the training examples with $k_{\dot\beta\tilde\alpha_1}$ and then uses that similarity to linearly weight the true function values from the training set $y_{i;\tilde\alpha_2}$ with the sample-space metric $\widetilde{k}^{\tilde\alpha_1\tilde\alpha_2}$. For this reason, kernel methods are sometimes referred to as *memory-based* methods since they involve memorizing the entire training set.[47] This should be contrasted with the parameterized linear model solution (10.124), where we forget the training set samples and instead just explicitly store the optimal parameter values $W^\star_{ij}$.

Finally, note that there's "no wiring" in the prediction: the z_i component of the prediction is entirely determined by the y_i component of the training examples. This

[46] In some situations, specifying and evaluating the kernel is much simpler than specifying and evaluating the feature functions. For instance, the *Gaussian kernel*, given by

$$k_{\delta_1\delta_2} \equiv \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n_0}\left(x_{i;\delta_1}-x_{i;\delta_2}\right)^2\right], \tag{10.133}$$

implies an infinite-dimensional feature space but can be evaluated by simply computing the squared distance between the n_{in}-dimensional input vectors and then exponentiating the result. (To see why the Gaussian kernel implies an infinite number of feature functions, we can express the squared distance as a sum of three inner products and then Taylor-expand the exponential in those inner products; the terms in the Taylor expansion give the feature functions.) In this way, we see how computing the kernel can be far easier than representing the features explicitly.

In fact, any algorithm based on a linear kernel (10.132) can be generalized by swapping the simple kernel for a more complicated kernel like the Gaussian kernel (10.133). This is known as the **kernel trick** and is a way to describe in the language of kernel methods how we generalized our simplest linear model (10.118) – that was linear in the input – to the more general linear model (10.119) – that was nonlinear in the input.

[47] Often, for a particular kernel method to be tractable, the model's predictions are made *locally*, incorporating information mostly from the training samples nearest to the test sample of interest. Given the $*$-polation results of last section, it's not hard to imagine how such methods could be made to work well.

A canonical example of such a local method is *k-nearest neighbors* [63, 64], which is a special type of kernel method. By only considering nearby training points, these kinds of local algorithms can skirt some of the impracticality that we've been pointing out for our exact Bayesian inference predictions as well as for our frozen NTK predictions. It would be interesting to extend such local algorithms to the finite-width exact Bayesian inference that we discussed in §6.4 or the finite-width gradient-based learning prediction that we'll discuss in §∞.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

is only implicit in the optimal weight matrix $W^\star_{ij}$ (10.123) in the linear model solution (10.124) but is explicit in the kernel method solution (10.131). This is one of many ways that such linear models and kernel methods are limited machine learning models.

10.4.3 Infinite-Width Networks as Linear Models

Surely, the kernel methods' prediction formula (10.131) should seem awfully familiar to you: it is precisely the same as the exact Bayesian mean prediction (6.64) if we identify the kernel methods' kernel $k_{\delta_1\delta_2}$ with the Bayesian kernel $K^{(L)}_{\delta_1\delta_2}$, and it is exactly the same as the (neural tangent) kernel mean prediction (10.40) if we identify the kernel methods' kernel $k_{\delta_1\delta_2}$ with the frozen neural tangent kernel $\Theta^{(L)}_{\delta_1\delta_2}$.[48] This finally provides a justification for the names of these objects as well as for the name of the current chapter.

Indeed, there is a very direct connection between these traditional linear models and kernel methods on the one hand and our (neural tangent) kernel learning of infinite-width models on the other hand. First, let's discuss the simpler case of the Bayesian kernel. As pointed out in §10.2.4, this choice corresponds to treating only output-layer biases $b^{(L)}_i$ and weights $W^{(L)}_{ij}$ as trainable model parameters, and so the network output at initialization is given by

$$z^{(L)}_i(x_\delta;\theta) = b^{(L)}_i + \sum_{j=1}^{n_{L-1}} W^{(L)}_{ij}\sigma^{(L-1)}_{j;\delta} = \sum_{j=0}^{n_{L-1}} W^{(L)}_{ij}\sigma^{(L-1)}_{j;\delta}, \tag{10.134}$$

where on the right-hand side we defined $\sigma^{(L-1)}_{0;\delta} \equiv 1$ and $W^{(L)}_{i0} \equiv b^{(L)}_i$. This is almost the same as our linear model (10.119) except that the feature functions are *random*: $\widehat{\phi}_{j;\delta} \equiv \sigma^{(L-1)}_{j;\delta}$. In particular, here we've *hatted* these feature functions to emphasize that they depend on the parameters in the hidden layers that are sampled from the initialization distribution at the beginning and then *fixed*; this is sometimes called a **random feature model**.

In this case, the kernel methods' notion of the kernel is also stochastic,

$$\begin{aligned}\widehat{k}_{\delta_1\delta_2} &= C^{(L)}_b\widehat{\phi}_{0;\delta_1}\widehat{\phi}_{0;\delta_2} + \frac{C^{(L)}_W}{n_{L-1}}\sum_{j=1}^{n_{L-1}}\widehat{\phi}_{j;\delta_1}\widehat{\phi}_{j;\delta_2} \\ &= C^{(L)}_b + C^{(L)}_W\left(\frac{1}{n_{L-1}}\sum_{j=1}^{n_{L-1}}\sigma^{(L-1)}_{j;\delta_1}\sigma^{(L-1)}_{j;\delta_2}\right),\end{aligned} \tag{10.135}$$

[48] You may have noticed that our stochastic (neural tangent) kernel prediction formula (10.39) also depended on the network output at initialization and had a nonzero covariance. This is related to our earlier discussion in footnote 44 that, when we're in the *overparameterized* regime with $(n_f+1) > N_{\mathcal{A}}$, as is especially the case when we have an infinite number of features, the (n_f+1)-by-(n_f+1) matrix matrix $M_{ij} \equiv \sum_{\tilde{\alpha}\in\mathcal{A}}\phi_i(x_{\tilde{\alpha}})\phi_j(x_{\tilde{\alpha}})$ (10.122) does not have a unique inverse. Thus, in this regime, the optimal weight matrix $W^\star$ is not unique: if we don't use the regulation trick (10.127) to uniquely pick out one of the solutions, the prediction in the dual kernel description will have a dependence on the model's initialization.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

where in the first line we have re-weighted the terms in the feature sum in the definition of the kernel (10.126) by $C_b^{(L)}$ and $C_W^{(L)}/n_L$.[49] Note that we called this object (10.135) the *stochastic metric* (4.70) when studying the RG flow of preactivations. Now, taking an expectation over the initialization ensemble, in the infinite-width limit we have

$$\mathbb{E}\left[\widehat{k}_{\delta_1\delta_2}\right] = C_b^{(L)} + C_W^{(L)} \left\langle \sigma_{\delta_1}\sigma_{\delta_2}\right\rangle_{K^{(L-1)}} = K^{(L)}_{\delta_1\delta_2}, \tag{10.136}$$

where in the last equality we used the recursion for the kernel, (10.44).[50] In this way, we see how we can interpret exact Bayesian inference at infinite width as a simple linear model (10.134) of fixed random features.

Now, let's give a linear model interpretation to gradient-based learning at infinite width. Since a linear model is linear in its parameters, we can more generally define the **random features** by

$$\widehat{\phi}_{i,\mu}(x_\delta) \equiv \frac{dz^{(L)}_{i;\delta}}{d\theta_\mu}. \tag{10.137}$$

To be clear, this derivative is evaluated at initialization, and these features are thus fixed in the infinite-width limit. Explicitly, for an MLP the random features are given by

$$\widehat{\phi}_{i,W^{(\ell)}_{k_1k_2}}(x_\delta) = \left(\sum_{j_{L-1},\ldots,j_{\ell+1}} W^{(L)}_{ij_{L-1}} \sigma'^{(L-1)}_{j_{L-1};\delta} \cdots W^{(\ell+1)}_{j_{\ell+1}k_1} \sigma'^{(\ell)}_{k_1;\delta}\right) \sigma^{(\ell-1)}_{k_2;\delta}, \tag{10.138}$$

with the bias component given by setting $\sigma^{(\ell)}_{0;\delta} \equiv 1$ and $W^{(\ell)}_{k0} \equiv b^{(\ell)}_k$. As is apparent from this expression, these features are stochastic, depending on the specific values of the biases and weights at initialization. Note also that the Bayesian linear model (10.134) only uses a subset of these features, $\widehat{\phi}_{i,W^{(L)}_{ij}} = \sigma^{(L-1)}_{j;\delta}$, and thus is a much more limited and less expressive model.

Note further that these feature functions $\widehat{\phi}_{i,\mu}(x_\delta)$ are related to, but not exactly equivalent to, the previous notion of feature we gave when we discussed representation group flow in §4.6. In that case, our ℓ-th-layer features corresponds to ℓ-th-layer preactivations $z^{(\ell)}_{i;\delta}$ or activations $\sigma^{(\ell)}_{i;\delta}$. However, here we see that the random feature

[49] A more general definition of the kernel methods' kernel (10.126) allows us to weight the contribution of each pair of feature functions as

$$k_{\delta_1\delta_2} \equiv \sum_{i,j=0}^{n_f} c_{ij}\, \phi_i(x_{\delta_1})\, \phi_j(x_{\delta_2})\,. \tag{10.139}$$

[50] Note alternatively, by the central limit theorem, that the stochastic kernel $\widehat{k}_{\delta_1\delta_2}$ will be equal to the kernel $K^{(L)}_{\delta_1\delta_2}$ in the infinite-width limit without explicitly averaging over initializations. This *self-averaging* of the kernel is equivalent to the fact that the connected four-point correlator vanishes at infinite width. Here we see that such self-averaging of the kernel can also be thought of as arising from a sum over an infinite number of random features.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

functions (10.138) are proportional to $(\ell-1)$-th-layer activations $\sigma_{i;\delta}^{(\ell-1)}$ but are also multiplied by objects from deeper layers.[51]

As should be clear, the stochastic kernel associated with these features,

$$\widehat{k}_{ij;\delta_1\delta_2} \equiv \sum_{\mu,\nu} \lambda_{\mu\nu}\, \widehat{\phi}_{i,\mu}(x_{\delta_1})\widehat{\phi}_{j,\nu}(x_{\delta_2}) = \sum_{\mu,\nu} \lambda_{\mu\nu} \frac{dz^{(L)}_{i;\delta_1}}{d\theta_\mu} \frac{dz^{(L)}_{j;\delta_2}}{d\theta_\nu} \equiv \widehat{H}^{(L)}_{ij;\delta_1\delta_2}, \tag{10.140}$$

is just the L-th-layer stochastic NTK (8.4).[52] Here, we have taken advantage of our more general definition of the kernel methods' kernel (10.139) to incorporate the learning-rate tensor $\lambda_{\mu\nu}$ into the expression. Accordingly, at infinite width the NTK is frozen and diagonal in final layer neural indices, giving

$$k_{\delta_1\delta_2} \equiv \sum_{\mu,\nu} \lambda_{\mu\nu}\, \widehat{\phi}_{i,\mu}(x_{\delta_1})\, \widehat{\phi}_{i,\nu}(x_{\delta_2}) = \sum_{\mu,\nu} \lambda_{\mu\nu} \frac{dz^{(L)}_{i;\delta_1}}{d\theta_\mu} \frac{dz^{(L)}_{i;\delta_2}}{d\theta_\nu} \equiv \Theta^{(L)}_{\delta_1\delta_2}. \tag{10.141}$$

In this way, we see that at infinite width the fully-trained mean network output is just a linear model based on random features (10.137). In this sense, infinite-width neural networks are rather shallow in terms of model complexity, however deep they may appear.

Looking back, when we discussed the linear model at the beginning of this section, we had to introduce feature functions $\phi_i(x)$, designed using our knowledge of the task and data at hand, as a way to (pre-)process the input. This way, the parametric model that we learn is really simple, i.e., linear.[53] We then reinterpreted this fit linear model in terms of its associated kernel, which itself has a natural interpretation as measuring similarity between our designed features.

However, for infinite-width networks we didn't *design* the frozen NTK, and its associated features are *random*. Instead, the network is defined by the architecture,

[51] Going forward, when referring to the *features* of a network, we will mean the kernel methods' notion of a feature function $\widehat{\phi}_{i,\mu}(x_\delta)$, rather than an activation $\sigma_i^{(\ell)}(x_\delta)$. Accordingly, we will now understand *representation learning* to describe how such feature functions develop a data dependence during training.

[52] However, please note importantly that at finite width the stochastic neural tangent *kernel* $\widehat{k}_{ij;\delta_1\delta_2} = \widehat{H}^{(L)}_{i_1 i_2;\delta_1\delta_2}$ is not fixed during training – learning useful features from the data – and randomly varies across initializations; hence – despite its name – it is not actually a kernel.

[53] Please don't confuse our discussion of *linear models* here and our discussion of linear vs. nonlinear functions as in §10.3.2.

A linear model is a model that's linear in the model parameters (10.119) and has a dual kernel description that's linear in the true outputs $y_{i;\tilde{\alpha}}$ in the training set $\mathcal{A}$ (10.131); as we have seen, linear models are very simple and easy to solve analytically. As is clear from the definition (10.119), a linear model in general will be *nonlinear* in the inputs x for general nonlinear feature functions $\phi_i(x)$. Accordingly, linear models can compute nonlinear functions of their input. We saw this explicitly when we worked out how nonlinear $*$-polation works for smooth nonlinear networks in (10.113).

In contrast, a deep linear network is a neural network that uses a `linear` activation function. Such networks compute linear functions of their input and thus may only linearly $*$-polate between training points, cf. (10.104). However, since they are *not* linear models (for $L > 1$), the function they compute depends nonlinearly on the model parameters. Accordingly, their training dynamics can be somewhat complicated, and at finite width they even exhibit representation learning.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press

hyperparameters, and biases and weights, and the path from those variables to the NTK and kernel prediction is filled with calculations. So the abstraction of the actual neural network seems like a very odd way to design a kernel.

Indeed, we've just learned that infinite-width neural networks can only make predictions that are linear in the true outputs from the training set; they are linear models that can only compute linear combinations of random features. Of course, deep learning is exciting because it works on problems where classic machine learning methods have failed; it works in cases where we don't know how to design feature functions or kernels, or doing so would be too complicated. For neural networks to go beyond the kernel methods, they need to be able to *learn* useful features *from the data*, not just make use of a complicated linear combination of random features. Thus, in order for the feature functions (10.137) to evolve during training – that is, in order to have *representation learning* – we will need to go beyond kernel methods.

Luckily, finite-width networks are not kernel methods. Please now turn to the next chapter to find out exactly what they are instead.

https://doi.org/10.1017/9781009023405.012 Published online by Cambridge University Press