Chapter 8

Measuring performance

Previous chapters described neural network models, loss functions, and training algorithms. This chapter considers how to measure the performance of the trained models. With sufficient capacity (i.e., number of hidden units), a neural network model will often perform perfectly on the training data. However, this does not necessarily mean it will generalize well to new test data.

We will see that the test errors have three distinct causes and that their relative contributions depend on (i) the inherent uncertainty in the task, (ii) the amount of training data, and (iii) the choice of model. The latter dependency raises the issue of hyperparameter search. We discuss how to select both the model hyperparameters (e.g., the number of hidden layers and the number of hidden units in each) and the learning algorithm hyperparameters (e.g., the learning rate and batch size).

8.1 Training a simple model

We explore model performance using the MNIST-1D dataset (figure 8.1). This consists of ten classes $y \in \{0, 1, \ldots, 9\}$, representing the digits 0–9. The data are derived from 1D templates for each of the digits. Each data example $\mathbf{x}$ is created by randomly transforming one of these templates and adding noise. The full training dataset $\{\mathbf{x}_i, y_i\}$ consists of $I = 4000$ training examples, each consisting of $D_i = 40$ dimensions representing the horizontal offset at 40 positions. The ten classes are drawn uniformly during data generation, so there are $\sim$400 examples of each class.

We use a network with $D_i = 40$ inputs and $D_o = 10$ outputs which are passed through a softmax function to produce class probabilities (see section 5.5). The network has two hidden layers with $D = 100$ hidden units each. It is trained using stochastic gradient descent with batch size 100 and learning rate 0.1 for 6000 steps (150 epochs) with a multiclass cross-entropy loss (equation 5.24). Figure 8.2 shows that the training error decreases as training proceeds. The training data are classified perfectly after about 4000 steps. The training loss also decreases, eventually approaching zero.

Problem 8.1

However, this doesn't imply that the classifier is perfect; the model might have mem-

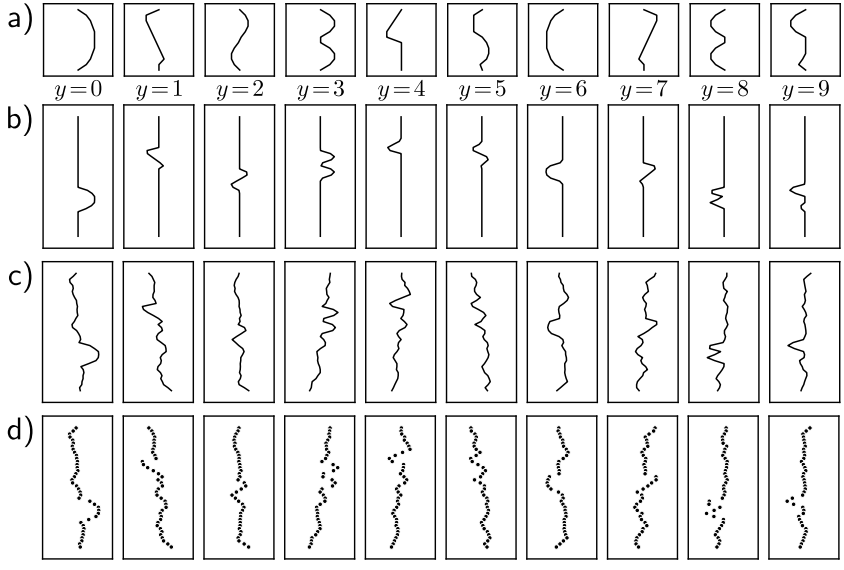


Figure 8.1 MNIST-1D. a) Templates for 10 classes $y \in \{0, \ldots, 9\}$, based on digits 0–9. b) Training examples $\mathbf{x}$ are created by randomly transforming a template and c) adding noise. d) The horizontal offset of the transformed template is then sampled at 40 vertical positions. Adapted from (Greydanus, 2020)

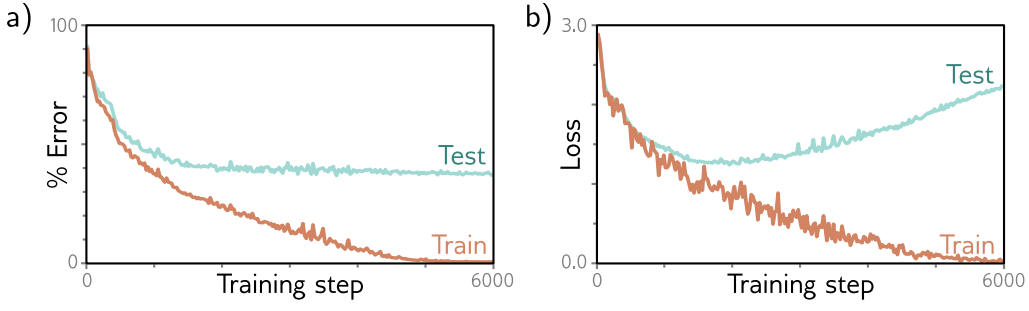


Figure 8.2 MNIST-1D results. a) Percent classification error as a function of the training step. The training set errors decrease to zero, but the test errors do not drop below $\sim 40\%$. This model doesn't generalize well to new test data. b) Loss as a function of the training step. The training loss decreases steadily toward zero. The test loss decreases at first but subsequently increases as the model becomes increasingly confident about its (wrong) predictions.

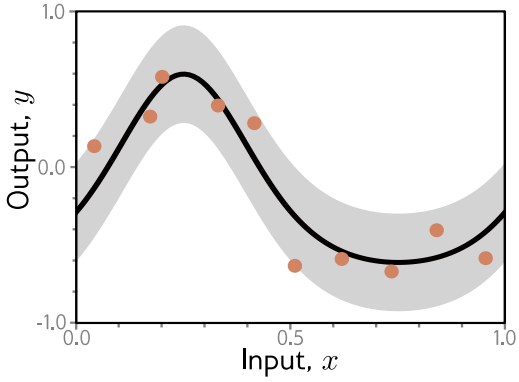


Figure 8.3 Regression function. Solid black line shows ground truth function. To generate I training examples $\{x_i, y_i\}$, the input space $x \in [0, 1]$ is divided into I equal segments and one sample x_i is drawn from a uniform distribution within each segment. The corresponding value y_i is created by evaluating the function at x_i and adding Gaussian noise (gray region shows ± 2 standard deviations). The test data are generated in the same way.

orized the training set but be unable to predict new examples. To estimate the true performance, we need a separate *test set* of input/output pairs $\{\mathbf{x}_i, y_i\}$. To this end, we generate 1000 more examples using the same process. Figure 8.2a also shows the errors for this test data as a function of the training step. These decrease as training proceeds, but only to around 40%. This is better than the chance error rate of 90% but far worse than for the training set; the model has not *generalized* well to the test data.

Notebook 8.1 MNIST-1D performance

The test loss (figure 8.2b) decreases for the first 1500 training steps but then increases again. At this point, the test error rate is fairly constant; the model makes the same mistakes but with increasing confidence. This decreases the probability of the correct answers and thus increases the negative log-likelihood. This increasing confidence is a side-effect of the softmax function; the pre-softmax activations are driven to increasingly extreme values to make the probability of the training data approach one (see figure 5.10).

8.2 Sources of error

We now consider the sources of the errors that occur when a model fails to generalize. To make this easier to visualize, we revert to a 1D linear least squares regression problem where we know exactly how the ground truth data were generated. Figure 8.3 shows a quasi-sinusoidal function; both training and test data are generated by sampling input values in the range $[0, 1]$, passing them through this function, and adding Gaussian noise with a fixed variance.

We fit a simplified shallow neural net to this data (figure 8.4). The weights and biases that connect the input layer to the hidden layer are chosen so that the "joints" of the function are evenly spaced across the interval. If there are D hidden units, then these joints will be at $0, 1/D, 2/D, \ldots, (D-1)/D$. This model can represent any piecewise linear function with D equally sized regions in the range $[0, 1]$. As well as being easy to understand, this model also has the advantage that it can be fit in closed form without the need for stochastic optimization algorithms (see problem 8.3). Consequently, we can guarantee to find the global minimum of the loss function during training.

Problems 8.2–8.3

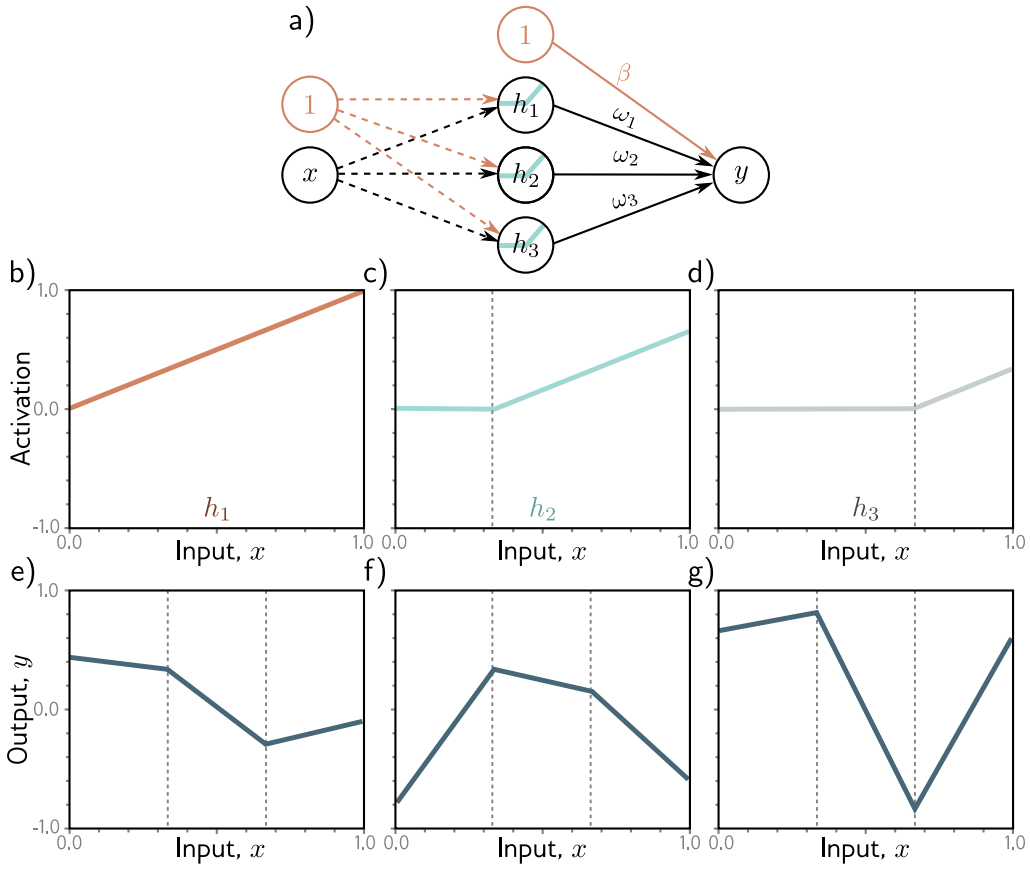


Figure 8.4 Simplified neural network with three hidden units. a) The weights and biases between the input and hidden layer are fixed (dashed arrows). b–d) They are chosen so that the hidden unit activations have slope one, and their joints are equally spaced across the interval, with joints at $x = 0$, $x = 1/3$, and $x = 2/3$, respectively. Modifying the remaining parameters $\boldsymbol{\phi} = \{\beta, \omega_1, \omega_2, \omega_3\}$ can create any piecewise linear function over $x \in [0, 1]$ with joints at 1/3 and 2/3. e–g) Three example functions with different values of the parameters $\boldsymbol{\phi}$.

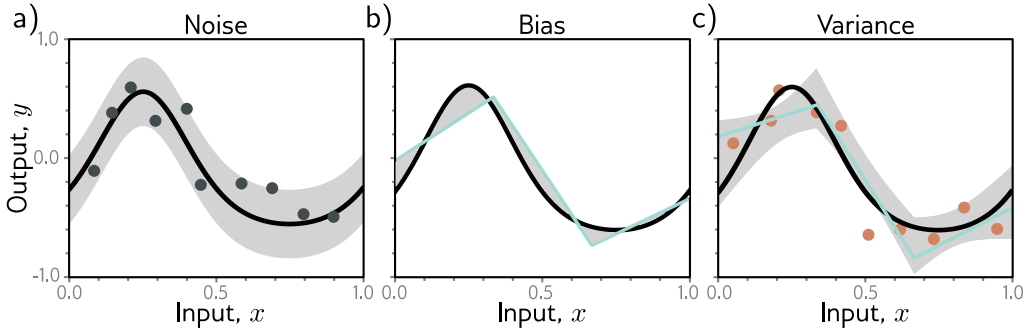


Figure 8.5 Sources of test error. a) Noise. Data generation is noisy, so even if the model exactly replicates the true underlying function (black line), the noise in the test data (gray points) means that some error will remain (gray region represents two standard deviations). b) Bias. Even with the best possible parameters, the three-region model (cyan line) cannot exactly fit the true function (black line). This bias is another source of error (gray regions represent signed error). c) Variance. In practice, we have limited noisy training data (orange points). When we fit the model, we don't recover the best possible function from panel (b) but a slightly different function (cyan line) that reflects idiosyncrasies of the training data. This provides an additional source of error (gray region represents two standard deviations). Figure 8.6 shows how this region was calculated.

8.2.1 Noise, bias, and variance

There are three possible sources of error, which are known as *noise*, *bias*, and *variance* respectively (figure 8.5):

Noise The data generation process includes the addition of noise, so there are multiple possible valid outputs y for each input x (figure 8.5a). This source of error is insurmountable for the test data. Note that it does not necessarily limit the training performance; we will likely never see the same input x twice during training, so it is still possible to fit the training data perfectly.

Noise may arise because there is a genuine stochastic element to the data generation process, because some of the data are mislabeled, or because there are further explanatory variables that were not observed. In rare cases, noise may be absent; for example, a network might approximate a function that is deterministic but requires significant computation to evaluate. However, noise is usually a fundamental limitation on the possible test performance.

Bias A second potential source of error may occur because the model is not flexible enough to fit the true function perfectly. For example, the three-region neural network model cannot exactly describe the quasi-sinusoidal function, even when the parameters are chosen optimally (figure 8.5b). This is known as *bias*.

Variance We have limited training examples, and there is no way to distinguish systematic changes in the underlying function from noise in the underlying data. When we fit a model, we do not get the closest possible approximation to the true underlying function. Indeed, for different training datasets, the result will be slightly different each time. This additional source of variability in the fitted function is termed *variance* (figure 8.5c). In practice, there might also be additional variance due to the stochastic learning algorithm, which does not necessarily converge to the same solution each time.

8.2.2 Mathematical formulation of test error

We now make the notions of noise, bias, and variance mathematically precise. Consider a 1D regression problem where the data generation process has additive noise with variance σ^2 (e.g., figure 8.3); we can observe different outputs y for the same input x, so for each x, there is a distribution $Pr(y|x)$ with expected value (mean) $\mu[x]$:

Appendix C.2 Expectation

$$\mu[x] = \mathbb{E}_y[y[x]] = \int y[x] Pr(y|x) dy, \tag{8.1}$$

and fixed noise $\sigma^2 = \mathbb{E}_y\left[(\mu[x] - y[x])^2\right]$. Here we have used the notation $y[x]$ to specify that we are considering the output y at a given input position x.

Now consider a least squares loss between the model prediction $\mathrm{f}[x, \boldsymbol{\phi}]$ at position x and the observed value $y[x]$ at that position:

$$\begin{aligned} L[x] &= \big(\mathrm{f}[x, \boldsymbol{\phi}] - y[x]\big)^2 \\ &= \Big(\big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big) + \big(\mu[x] - y[x]\big)\Big)^2 \\ &= \big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)^2 + 2\big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)\big(\mu[x] - y[x]\big) + \big(\mu[x] - y[x]\big)^2, \end{aligned} \tag{8.2}$$

where we have both added and subtracted the mean $\mu[x]$ of the underlying function in the second line and have expanded out the squared term in the third line.

The underlying function is stochastic, so this loss depends on the particular $y[x]$ we observe. The expected loss is:

$$\begin{aligned} \mathbb{E}_y\big[L[x]\big] &= \mathbb{E}_y\Big[\big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)^2 + 2\big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)\big(\mu[x] - y[x]\big) + \big(\mu[x] - y[x]\big)^2\Big] \\ &= \big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)^2 + 2\big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)\big(\mu[x] - \mathbb{E}_y\left[y[x]\right]\big) + \mathbb{E}_y\left[(\mu[x] - y[x])^2\right] \\ &= \big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)^2 + 2\big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big) \cdot 0 + \mathbb{E}_y\left[\big(\mu[x] - y[x]\big)^2\right] \\ &= \big(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\big)^2 + \sigma^2, \end{aligned} \tag{8.3}$$

where we have made use of the rules for manipulating expectations. In the second line, we have distributed the expectation operator and removed it from terms with no dependence on $y[x]$, and in the third line, we note that the second term is zero since $\mathbb{E}_y[y[x]] = \mu[x]$ by definition. Finally, in the fourth line, we have substituted in the definition of the

Appendix C.2.1 Expectation rules

noise σ^2. We can see that the expected loss has been broken down into two terms; the first term is the squared deviation between the model and the true function mean, and the second term is the noise.

The first term can be further partitioned into bias and variance. The parameters $\boldsymbol{\phi}$ of the model $\mathrm{f}[x, \boldsymbol{\phi}]$ depend on the training dataset $\mathcal{D} = \{x_i, y_i\}$, so more properly, we should write $\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]]$. The training dataset is a random sample from the data generation process; with a different sample of training data, we would learn different parameter values. The expected model output $\mathrm{f}_\mu[x]$ with respect to all possible datasets $\mathcal{D}$ is hence:

$$\mathrm{f}_\mu[x] = \mathbb{E}_{\mathcal{D}}\Big[\mathrm{f}\big[x, \boldsymbol{\phi}[\mathcal{D}]\big]\Big]. \tag{8.4}$$

Returning to the first term of equation 8.3, we add and subtract $\mathrm{f}_\mu[x]$ and expand:

$$\begin{aligned}
&\big(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - \mu[x]\big)^2 \qquad (8.5)\\
&\quad = \Big(\big(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - \mathrm{f}_\mu[x]\big) + \big(\mathrm{f}_\mu[x] - \mu[x]\big)\Big)^2\\
&\quad = \big(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - \mathrm{f}_\mu[x]\big)^2 + 2\big(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - \mathrm{f}_\mu[x]\big)\big(\mathrm{f}_\mu[x] - \mu[x]\big) + \big(\mathrm{f}_\mu[x] - \mu[x]\big)^2.
\end{aligned}$$

We then take the expectation with respect to the training dataset $\mathcal{D}$:

$$\mathbb{E}_{\mathcal{D}}\Big[\big(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - \mu[x]\big)^2\Big] = \mathbb{E}_{\mathcal{D}}\Big[\big(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - \mathrm{f}_\mu[x]\big)^2\Big] + \big(\mathrm{f}_\mu[x] - \mu[x]\big)^2, \tag{8.6}$$

where we have simplified using similar steps as for equation 8.3. Finally, we substitute this result into equation 8.3:

$$\mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_y[L[x]]\Big] = \underbrace{\mathbb{E}_{\mathcal{D}}\Big[\big(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - \mathrm{f}_\mu[x]\big)^2\Big]}_{\text{variance}} + \underbrace{\big(\mathrm{f}_\mu[x] - \mu[x]\big)^2}_{\text{bias}} + \underbrace{\sigma^2}_{\text{noise}}. \tag{8.7}$$

This equation says that the expected loss after considering the uncertainty in the training data $\mathcal{D}$ and the test data y consists of three additive components. The variance is uncertainty in the fitted model due to the particular training dataset we sample. The bias is the systematic deviation of the model from the mean of the function we are modeling. The noise is the inherent uncertainty in the true mapping from input to output. These three sources of error will be present for any task. They combine additively for linear regression with a least squares loss. However, their interaction can be more complex for other types of problems.

8.3 Reducing error

In the previous section, we saw that test error results from three sources: noise, bias, and variance. The noise component is insurmountable; there is nothing we can do to circumvent this, and it represents a fundamental limit on expected model performance. However, it is possible to reduce the other two terms.

8.3.1 Reducing variance

Recall that the variance results from limited noisy training data. Fitting the model to two different training sets results in slightly different parameters. It follows we can reduce the variance by increasing the quantity of training data. This averages out the inherent noise and ensures that the input space is well sampled.

Figure 8.6 shows the effect of training with 6, 10, and 100 samples. For each dataset size, we show the best-fitting model for three training datasets. With only six samples, the fitted function is quite different each time: the variance is significant. As we increase the number of samples, the fitted models become very similar, and the variance reduces. In general, adding training data almost always improves test performance.

8.3.2 Reducing bias

The bias term results from the inability of the model to describe the true underlying function. This suggests that we can reduce this error by making the model more flexible. This is usually done by increasing the model *capacity*. For neural networks, this means adding more hidden units and/or hidden layers.

In the simplified model, adding capacity corresponds to adding more hidden units so that the interval $[0, 1]$ is divided into more linear regions. Figures 8.7a–c show that (unsurprisingly) this does indeed reduce the bias; as we increase the number of linear regions to ten, the model becomes flexible enough to fit the true function closely.

8.3.3 Bias-variance trade-off

However, figures 8.7d–f show an unexpected side-effect of increasing the model capacity. For a fixed-size training dataset, the variance term typically increases as the model capacity increases. Consequently, increasing the model capacity does not necessarily reduce the test error. This is known as the *bias-variance trade-off*.

Figure 8.8 explores this phenomenon. In panels a–c), we fit the simplified three-region model to three different datasets of fifteen points. Although the datasets differ, the final model is much the same; the noise in the dataset roughly averages out in each linear region. In panels d–f), we fit a model with ten regions to the same three datasets. This model has more flexibility, but this is disadvantageous; the model certainly fits the data better, and the training error will be lower, but much of the extra descriptive power is devoted to modeling the noise. This phenomenon is known as *overfitting*.

We've seen that as we add capacity to the model, the bias decreases, but the variance increases for a fixed-size training dataset. This suggests that there is an optimal capacity where the bias is not too large and the variance is still relatively small. Figure 8.9 shows how these terms vary numerically for the toy model as we increase the capacity, using the data from figure 8.8. For regression models, the total expected error is the sum of the bias and the variance, and this sum is minimized when the model capacity is four (i.e., with four hidden units and four linear regions in the range of the data).

Notebook 8.2
Bias-variance
trade-off

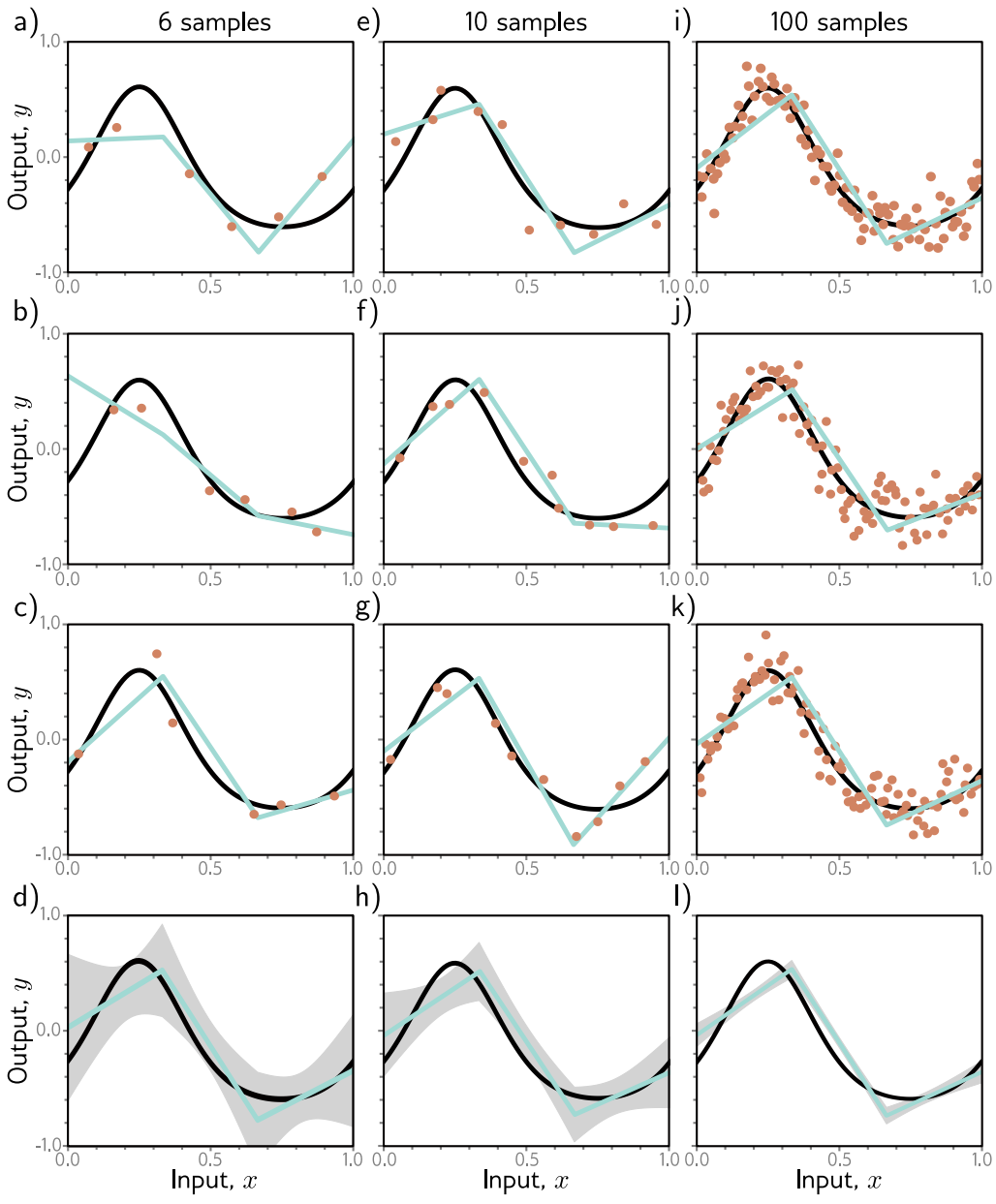


Figure 8.6 Reducing variance by increasing training data. a–c) The three-region model fitted to three different randomly sampled datasets of six points. The fitted model is quite different each time. d) We repeat this experiment many times and plot the mean model predictions (cyan line) and the variance of the model predictions (gray area shows two standard deviations). e–h) We do the same experiment, but this time with datasets of size ten. The variance of the predictions is reduced. i–l) We repeat this experiment with datasets of size 100. Now the fitted model is always similar, and the variance is small.

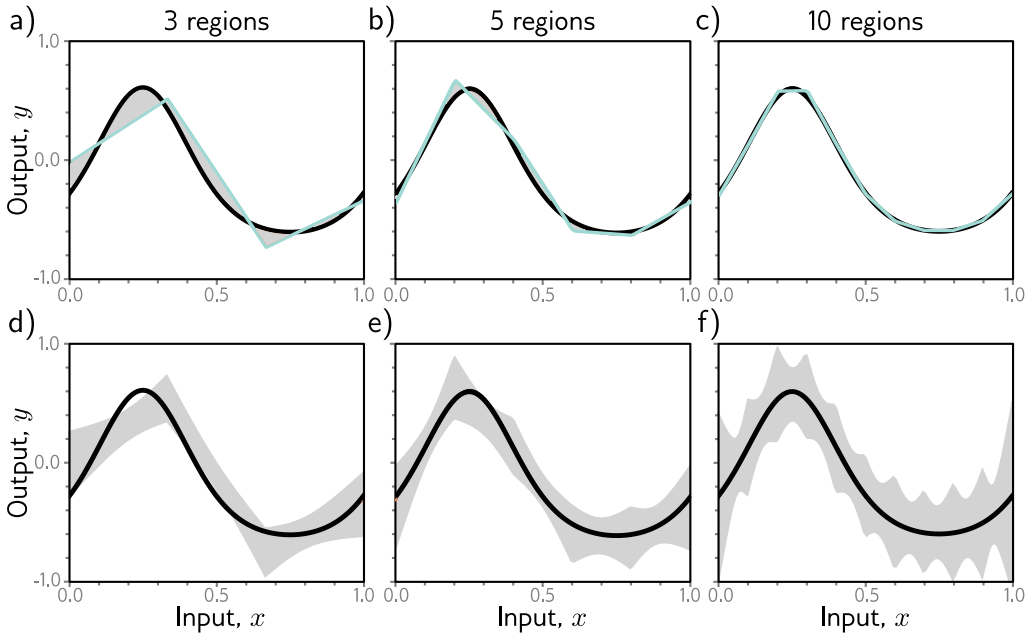


Figure 8.7 Bias and variance as a function of model capacity. a–c) As we increase the number of hidden units of the toy model, the number of linear regions increases, and the model becomes able to fit the true function closely; the bias (gray region) decreases. d–f) Unfortunately, increasing the model capacity has the side-effect of increasing the variance term (gray region). This is known as the bias-variance trade-off.

8.4 Double descent

In the previous section, we examined the bias-variance trade-off as we increased the capacity of a model. Let's now return to the MNIST-1D dataset and see whether this happens in practice. We use 10,000 training examples, test with another 5,000 examples and examine the training and test performance as we increase the capacity (number of parameters) in the model. We train the model with Adam and a step size of 0.005 using a full batch of 10,000 examples for 4000 steps.

Figure 8.10a shows the training and test error for a neural network with two hidden layers as the number of hidden units increases. The training error decreases as the capacity grows and quickly becomes close to zero. The vertical dashed line represents the capacity where the model has the same number of parameters as there are training examples, but the model memorizes the dataset before this point. The test error decreases as we add model capacity but does not increase as predicted by the bias-variance trade-off curve; it keeps decreasing.

In figure 8.10b, we repeat this experiment, but this time, we randomize 15% of the

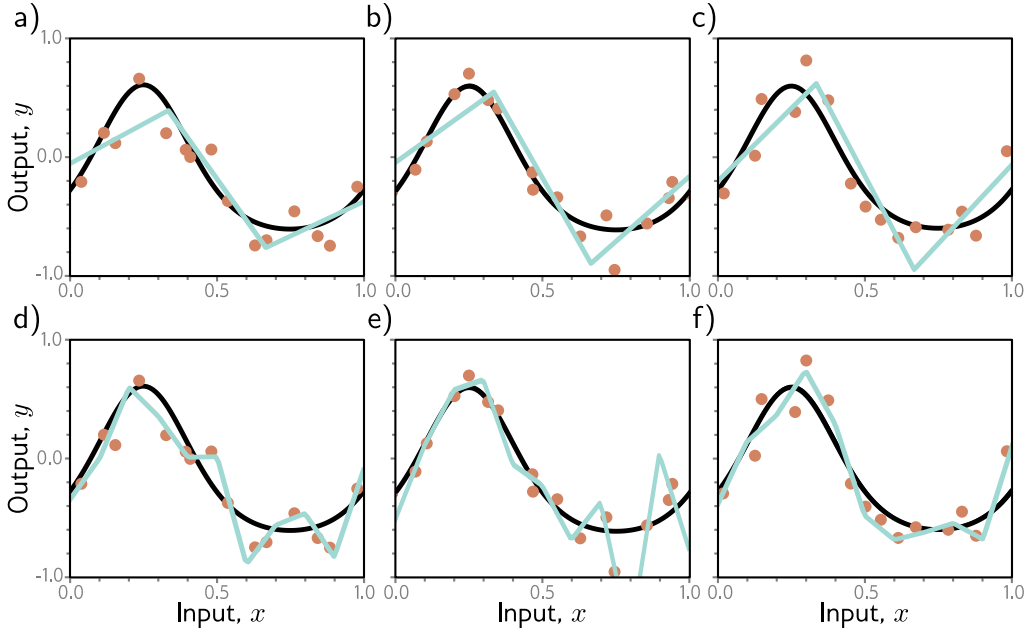
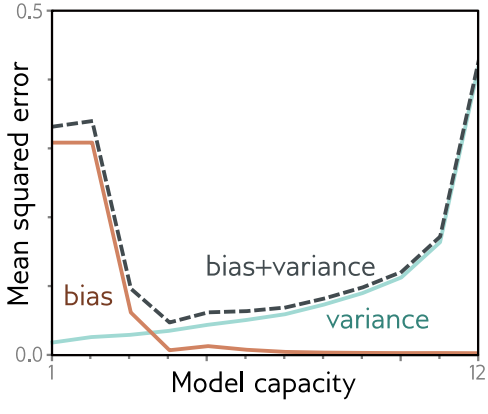


Figure 8.8 Overfitting. a–c) A model with three regions is fit to three different datasets of fifteen points each. The result is similar in all three cases (i.e., the variance is low). d–f) A model with ten regions is fit to the same datasets. The additional flexibility does not necessarily produce better predictions. While these three models each describe the training data better, they are not necessarily closer to the true underlying function (black curve). Instead, they overfit the data and describe the noise, and the variance (difference between fitted curves) is larger.

Figure 8.9 Bias-variance trade-off. The bias and variance terms from equation 8.7 are plotted as a function of the model capacity (number of hidden units / linear regions in range of data) in the simplified model using training data from figure 8.8. As the capacity increases, the bias (solid orange line) decreases, but the variance (solid cyan line) increases. The sum of these two terms (dashed gray line) is minimized when the capacity is four.



training labels. Once more, the training error decreases to zero. This time, there is more randomness, and the model requires almost as many parameters as there are data points to memorize the data. The test error does show the typical bias-variance trade-off as we increase the capacity to the point where the model fits the training data exactly. However, then it does something unexpected; it starts to decrease again. Indeed, if we add enough capacity, the test loss reduces to below the minimal level that we achieved in the first part of the curve.

This phenomenon is known as *double descent*. For some datasets like MNIST, it is present with the original data (figure 8.10c). For others, like MNIST-1D and CIFAR-100 (figure 8.10d), it emerges or becomes more prominent when we add noise to the labels. The first part of the curve is referred to as the *classical* or *under-parameterized regime*, and the second part as the *modern* or *over-parameterized regime*. The central part where the error increases is termed the *critical regime*.

Notebook 8.3
Double descent

8.4.1 Explanation

The discovery of double descent is recent, unexpected, and somewhat puzzling. It results from an interaction of two phenomena. First, the test performance becomes temporarily worse when the model has just enough capacity to memorize the data. Second, the test performance continues to improve with capacity even after the training performance is perfect. The first phenomenon is exactly as predicted by the bias-variance trade-off. The second phenomenon is more confusing; it's unclear why performance should be better in the over-parameterized regime, given that there are now not even enough training data points to constrain the model parameters uniquely.

To understand why performance continues to improve as we add more parameters, note that once the model has enough capacity to drive the training loss to near zero, the model fits the training data almost perfectly. This implies that further capacity cannot help the model fit the training data any better; any change must occur *between* the training points. The tendency of a model to prioritize one solution over another as it extrapolates between data points is known as its *inductive bias*.

Problems 8.4–8.5

The model's behavior between data points is critical because, in high-dimensional space, the training data are extremely sparse. The MNIST-1D dataset has 40 dimensions, and we trained with 10,000 examples. If this seems like plenty of data, consider what would happen if we quantized each input dimension into 10 bins. There would be 10^{40} bins in total, constrained by only 10^4 examples. Even with this coarse quantization, there will only be one data point in every 10^{35} bins! The tendency of the volume of high-dimensional space to overwhelm the number of training points is termed the *curse of dimensionality*.

The implication is that problems in high dimensions might look more like figure 8.11a; there are small regions of the input space where we observe data with significant gaps between them. The putative explanation for double descent is that as we add capacity to the model, it interpolates between the nearest data points increasingly smoothly. In the absence of information about what happens between the training points, assuming smoothness is sensible and will probably generalize reasonably to new data.

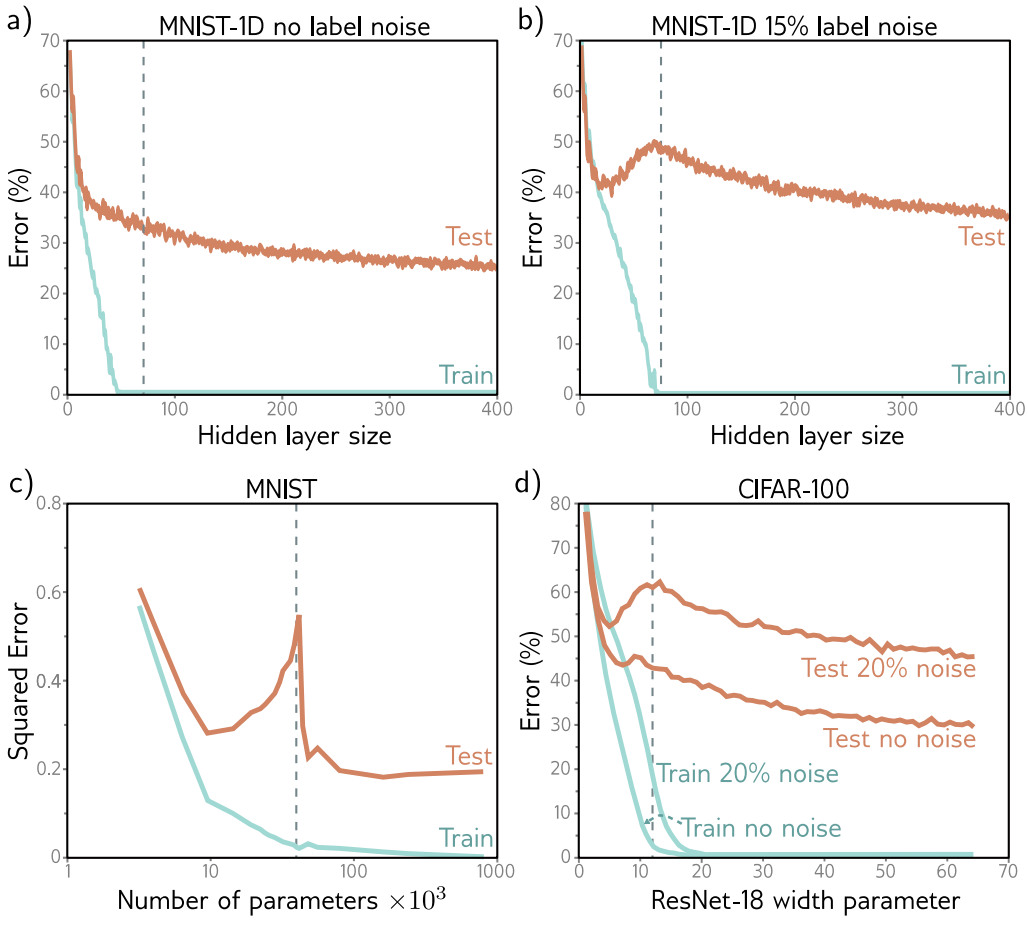


Figure 8.10 Double descent. a) Training and test loss on MNIST-1D for a two-hidden layer network as we increase the number of hidden units (and hence parameters) in each layer. The training loss decreases to zero as the number of parameters approaches the number of training examples (vertical dashed line). The test error does not show the expected bias-variance trade-off but continues to decrease even after the model has memorized the dataset. b) The same experiment is repeated with noisier training data. Again, the training error reduces to zero, although it now takes almost as many parameters as training points to memorize the dataset. The test error shows the predicted bias/variance trade-off; it decreases as the capacity increases but then increases again as we near the point where the training data is exactly memorized. However, it subsequently decreases again and ultimately reaches a better performance level. This is known as double descent. Depending on the loss, the model, and the amount of noise in the data, the double descent pattern can be seen to a greater or lesser degree across many datasets. c) Results on MNIST (without label noise) with shallow neural network from Belkin et al. (2019). d) Results on CIFAR-100 with ResNet18 network (see chapter 11) from Nakkiran et al. (2021). See original papers for details.

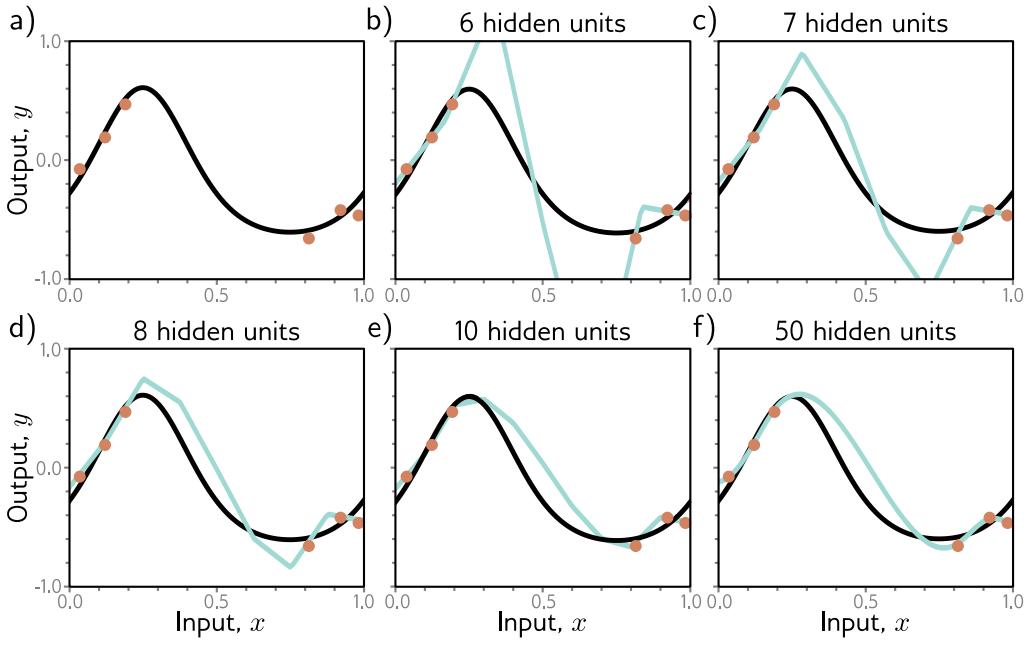


Figure 8.11 Increasing capacity (hidden units) allows smoother interpolation between sparse data points. a) Consider this situation where the training data (orange circles) are sparse; there is a large region in the center with no data examples to constrain the model to mimic the true function (black curve). b) If we fit a model with just enough capacity to fit the training data (cyan curve), then it has to contort itself to pass through the training data, and the output predictions will not be smooth. c–f) However, as we add more hidden units, the model has the *ability* to interpolate between the points more smoothly (smoothest possible curve plotted in each case). However, unlike in this figure, it is not obliged to.

This argument is plausible. It's certainly true that as we add more capacity to the model, it will have the capability to create smoother functions. Figures 8.11b–f show the smoothest possible functions that still pass through the data points as we increase the number of hidden units. When the number of parameters is very close to the number of training data examples (figure 8.11b), the model is forced to contort itself to fit the training data exactly, resulting in erratic predictions. This explains why the peak in the double descent curve is so pronounced. As we add more hidden units, the model has the ability to construct smoother functions that are likely to generalize better to new data.

However, this does not explain *why* over-parameterized models should produce smooth functions. Figure 8.12 shows three functions that can be created by the simplified model with 50 hidden units. In each case, the model fits the data exactly, so the loss is zero. If the modern regime of double descent is explained by increasing smoothness, then what exactly is encouraging this smoothness?

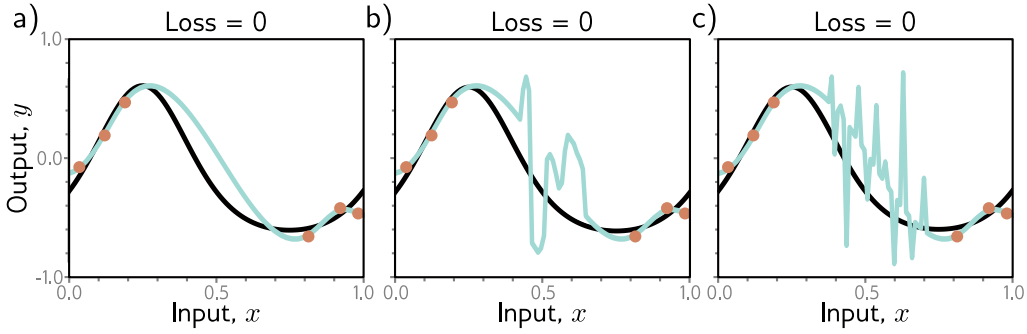


Figure 8.12 Regularization. a–c) Each of the three fitted curves passes through the data points exactly, so the training loss for each is zero. However, we might expect the smooth curve in panel (a) to generalize much better to new data than the erratic curves in panels (b) and (c). Any factor that biases a model toward a subset of the solutions with a similar training loss is known as a regularizer. It is thought that the initialization and/or fitting of neural networks have an implicit regularizing effect. Consequently, in the over-parameterized regime, more reasonable solutions, such as that in panel (a), are encouraged.

The answer to this question is uncertain, but there are two likely possibilities. First, the network initialization may encourage smoothness, and the model never departs from the sub-domain of smooth function during the training process. Second, the training algorithm may somehow "prefer" to converge to smooth functions. Any factor that biases a solution toward a subset of equivalent solutions is known as a *regularizer*, so one possibility is that the training algorithm acts as an implicit regularizer (see section 9.2).

8.5 Choosing hyperparameters

In the previous section, we discussed how test performance changes with model capacity. Unfortunately, in the classical regime, we don't have access to either the bias (which requires knowledge of the true underlying function) or the variance (which requires multiple independently sampled datasets to estimate). In the modern regime, there is no way to tell how much capacity should be added before the test error stops improving. This raises the question of exactly how we should choose model capacity in practice.

For a deep network, the model capacity depends on the numbers of hidden layers and hidden units per layer as well as other aspects of architecture that we have yet to introduce. Furthermore, the choice of learning algorithm and any associated parameters (learning rate, etc.) also affects the test performance. These elements are collectively termed *hyperparameters*. The process of finding the best hyperparameters is termed *hyperparameter search* or (when focused on network structure) *neural architecture search*.

Hyperparameters are typically chosen empirically; we train many models with different hyperparameters on the same training set, measure their performance, and retain the best model. However, we do not measure their performance on the test set; this would admit the possibility that these hyperparameters just happen to work well for the test set but don't generalize to further data. Instead, we introduce a third dataset known as a *validation set*. For every choice of hyperparameters, we train the associated model using the training set and evaluate performance on the validation set. Finally, we select the model that worked best on the validation set and measure its performance on the test set. In principle, this should give a reasonable estimate of the true performance.

The hyperparameter space is generally smaller than the parameter space but still too large to try every combination exhaustively. Unfortunately, many hyperparameters are discrete (e.g., the number of hidden layers), and others may be conditional on one another (e.g., we only need to specify the number of hidden units in the tenth hidden layer if there are ten or more layers). Hence, we cannot rely on gradient descent methods as we did for learning the model parameters. Hyperparameter optimization algorithms intelligently sample the space of hyperparameters, contingent on previous results. This procedure is computationally expensive since we must train an entire model and measure the validation performance for each combination of hyperparameters.

8.6 Summary

To measure performance, we use a separate test set. The degree to which performance is maintained on this test set is known as generalization. Test errors can be explained by three factors: noise, bias, and variance. These combine additively in regression problems with least squares losses. Adding training data decreases the variance. When the model capacity is less than the number of training examples, increasing the capacity decreases bias but increases variance. This is known as the bias-variance trade-off, and there is a capacity where the trade-off is optimal.

However, this is balanced against a tendency for performance to improve with capacity, even when the parameters exceed the training examples. Together, these two phenomena create the double descent curve. It is thought that the model interpolates more smoothly between the training data points in the over-parameterized "modern regime," although it is unclear what drives this. To choose the capacity and other model and training algorithm hyperparameters, we fit multiple models and evaluate their performance using a separate validation set.

Notes

Bias-variance trade-off: We showed that the test error for regression problems with least squares loss decomposes into the sum of noise, bias, and variance terms. These factors are all present for models with other losses, but their interaction is typically more complicated (Friedman, 1997; Domingos, 2000). For classification problems, there are some counter-intuitive

predictions; for example, if the model is biased toward selecting the wrong class in a region of the input space, then increasing the variance can improve the classification rate as this pushes some of the predictions over the threshold to be classified correctly.

Cross-validation: We saw that it is typical to divide the data into three parts: training data (which is used to learn the model parameters), validation data (which is used to choose the hyperparameters), and test data (which is used to estimate the final performance). This approach is known as *cross-validation*. However, this division may cause problems where the total number of data examples is limited; if the number of training examples is comparable to the model capacity, then the variance will be large.

One way to mitigate this problem is to use *k-fold cross-validation*. The training and validation data are partitioned into K disjoint subsets. For example, we might divide these data into five parts. We train with four and validate with the fifth for each of the five permutations and choose the hyperparameters based on the average validation performance. The final test performance is assessed using the average of the predictions from the five models with the best hyperparameters on an entirely different test set. There are many variations of this idea, but all share the general goal of using a larger proportion of the data to train the model, thereby reducing variance.

Capacity: We have used the term *capacity* informally to mean the number of parameters or hidden units in the model (and hence indirectly, the ability of the model to fit functions of increasing complexity). The *representational capacity* of a model describes the space of possible functions it can construct when we consider all possible parameter values. When we take into account the fact that an optimization algorithm may not be able to reach all of these solutions, what is left is the *effective capacity*.

The Vapnik-Chervonenkis (VC) dimension (Vapnik & Chervonenkis, 1971) is a more formal measure of capacity. It is the largest number of training examples that a binary classifier can label arbitrarily. Bartlett et al. (2019) derive upper and lower bounds for the VC dimension in terms of the number of layers and weights. An alternative measure of capacity is the Rademacher complexity, which is the expected empirical performance of a classification model (with optimal parameters) for data with random labels. Neyshabur et al. (2017) derive a lower bound on the generalization error in terms of the Rademacher complexity.

Double descent: The term "double descent" was coined by Belkin et al. (2019), who demonstrated that the test error decreases again in the over-parameterized regime for two-layer neural networks and random features. They also claimed that this occurs in decision trees, although Buschjäger & Morik (2021) subsequently provided evidence to the contrary. Nakkiran et al. (2021) show that double descent occurs for various modern datasets (CIFAR-10, CIFAR-100, IWSLT'14 de-en), architectures (CNNs, ResNets, transformers), and optimizers (SGD, Adam). The phenomenon is more pronounced when noise is added to the target labels (Nakkiran et al., 2021) and when some regularization techniques are used (Ishida et al., 2020).

Nakkiran et al. (2021) also provide empirical evidence that test performance depends on *effective model capacity* (the largest number of samples for which a given model and training method can achieve zero training error). At this point, the model starts to devote its efforts to interpolating smoothly. As such, the test performance depends not just on the model but also on the training algorithm and length of training. They observe the same pattern when they study a model with fixed capacity and increase the number of training iterations. They term this *epoch-wise double descent*. This phenomenon has been modeled by Pezeshki et al. (2022) in terms of different features in the model being learned at different speeds.

Double descent makes the rather strange prediction that adding training data can sometimes worsen test performance. Consider an over-parameterized model in the second descending part

of the curve. If we increase the training data to match the model capacity, we will now be in the critical region of the new test error curve, and the test loss may increase.

Bubeck & Sellke (2021) prove that overparameterization is necessary to interpolate data smoothly in high dimensions. They demonstrate a trade-off between the number of parameters and the Lipschitz constant of a model (the fastest the output can change for a small input change). A review of the theory of over-parameterized machine learning can be found in Dar et al. (2021).

Appendix B.1.1 Lipschitz constant

Curse of dimensionality: As dimensionality increases, the volume of space grows so fast that the amount of data needed to densely sample it increases exponentially. This phenomenon is known as the curse of dimensionality. High-dimensional space has many unexpected properties, and caution should be used when trying to reason about it based on low-dimensional examples. This book visualizes many aspects of deep learning in one or two dimensions, but these visualizations should be treated with healthy skepticism.

Surprising properties of high-dimensional spaces include: (i) Two randomly sampled data points from a standard normal distribution are very close to orthogonal to one another (relative to the origin) with high likelihood. (ii) The distance from the origin of samples from a standard normal distribution is roughly constant. (iii) Most of a volume of a high-dimensional sphere (hypersphere) is adjacent to its surface (a common metaphor is that most of the volume of a high-dimensional orange is in the peel, not in the pulp). (iv) If we place a unit-diameter hypersphere inside a hypercube with unit-length sides, then the hypersphere takes up a decreasing proportion of the volume of the cube as the dimension increases. Since the volume of the cube is fixed at size one, this implies that the volume of a high-dimensional hypersphere becomes close to zero. (v) For random points drawn from a uniform distribution in a high-dimensional hypercube, the ratio of the Euclidean distance between the nearest and furthest points becomes close to one. For further information, consult Beyer et al. (1999) and Aggarwal et al. (2001).

Problems 8.6–8.9

Notebook 8.4 High-dimensional spaces

Real-world performance: In this chapter, we argued that model performance could be evaluated using a held-out test set. However, the result won't be indicative of real-world performance if the statistics of the test set don't match those of real-world data. Moreover, the statistics of real-world data may change over time, causing the model to become increasingly stale and performance to decrease. This is known as *data drift* and means that deployed models must be carefully monitored.

There are three main reasons why real-world performance may be worse than the test performance implies. First, the statistics of the input data $\mathbf{x}$ may change; we may now be observing parts of the function that were sparsely sampled or not sampled at all during training. This is known as *covariate shift*. Second, the statistics of the output data $\mathbf{y}$ may change; if some output values are infrequent during training, then the model may learn not to predict these in ambiguous situations and will make mistakes if they are more common in the real world. This is known as *prior shift*. Third, the relationship between input and output may change. This is known as *concept shift*. These issues are discussed in Moreno-Torres et al. (2012).

Hyperparameter search: Finding the best hyperparameters is a challenging optimization task. Testing a single configuration of hyperparameters is expensive; we must train an entire model and measure its performance. We have no easy way to access the derivatives (i.e., how performance changes when we make a small change to a hyperparameter). Moreover, many of the hyperparameters are discrete, so we cannot use gradient descent methods. There are multiple local minima and no way to tell if we are close to the global minimum. The noise level is high since each training/validation cycle uses a stochastic training algorithm; we expect different results if we train a model twice with the same hyperparameters. Finally, some variables are conditional and only exist if others are set. For example, the number of hidden units in the third hidden layer is only relevant if we have at least three hidden layers.

A simple approach is to sample the space randomly (Bergstra & Bengio, 2012). However, for continuous variables, it is better to build a model of performance as a function of the hyperparameters and the uncertainty in this function. This can be exploited to test where the uncertainty is great (explore the space) or home in on regions where performance looks promising (exploit previous knowledge). Bayesian optimization is a framework based on Gaussian processes that does just this, and its application to hyperparameter search is described in Snoek et al. (2012). The Beta-Bernoulli bandit (see Lattimore & Szepesvári, 2020) is a roughly equivalent model for describing uncertainty in results due to discrete variables.

The sequential model-based configuration (SMAC) algorithm (Hutter et al., 2011) can cope with continuous, discrete, and conditional parameters. The basic approach is to use a random forest to model the objective function where the mean of the tree predictions is the best guess about the objective function, and their variance represents the uncertainty. A completely different approach that can also cope with combinations of continuous, discrete, and conditional parameters is Tree-Parzen Estimators (Bergstra et al., 2011). The previous methods modeled the probability of the model performance given the hyperparameters. In contrast, the Tree-Parzen estimator models the probability of the hyperparameters given the model performance.

Hyperband (Li et al., 2017b) is a multi-armed bandit strategy for hyperparameter optimization. It assumes that there are computationally cheap but approximate ways to measure performance (e.g., by not training to completion) and that these can be associated with a budget (e.g., by training for a fixed number of iterations). A number of random configurations are sampled and run until the budget is used up. Then the best fraction η of runs is kept, and the budget is multiplied by $1/\eta$. This is repeated until the maximum budget is reached. This approach has the advantage of efficiency; for bad configurations, it does not need to run the experiment to the end. However, each sample is just chosen randomly, which is inefficient. The BOHB algorithm (Falkner et al., 2018) combines the efficiency of Hyperband with the more sensible choice of hyperparameters from Tree Parzen estimators to construct an even better method.

Problems

Problem 8.1 Will the multiclass cross-entropy training loss in figure 8.2 ever reach zero? Explain your reasoning.

Problem 8.2 What values should we choose for the three weights and biases in the first layer of the model in figure 8.4a so that the hidden unit's responses are as depicted in figures 8.4b–d?

Problem 8.3* Given a training dataset consisting of I input/output pairs $\{x_i, y_i\}$, show how the parameters $\{\beta, \omega_1, \omega_2, \omega_3\}$ for the model in figure 8.4a using the least squares loss function can be found in closed form.

Problem 8.4 Consider the curve in figure 8.10b at the point where we train a model with a hidden layer of size 200, which would have 50,410 parameters. What do you predict will happen to the training and test performance if we increase the number of training examples from 10,000 to 50,410?

Problem 8.5 Consider the case where the model capacity exceeds the number of training data points, and the model is flexible enough to reduce the training loss to zero. What are the implications of this for fitting a heteroscedastic model? Propose a method to resolve any problems that you identify.

Problem 8.6 Show that two random points drawn from a 1000-dimensional standard Gaussian distribution are orthogonal relative to the origin with high probability.

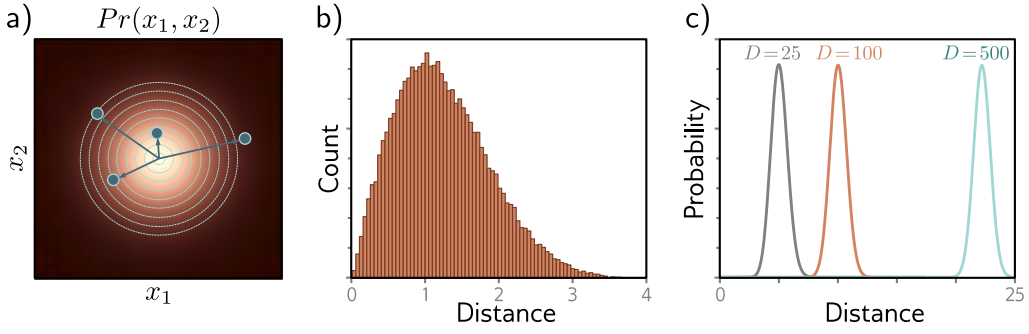


Figure 8.13 Typical sets. a) Standard normal distribution in two dimensions. Circles are four samples from this distribution. As the distance from the center increases, the probability decreases, but the volume of space at that radius (i.e., the area between adjacent evenly spaced circles) increases. b) These factors trade off so that the histogram of distances of samples from the center has a pronounced peak. c) In higher dimensions, this effect becomes more extreme, and the probability of observing a sample close to the mean becomes vanishingly small. Although the most likely point is at the mean of the distribution, the *typical samples* are found in a relatively narrow shell.

Problem 8.7 The volume of a hypersphere with radius r in D dimensions is:

$$\text{Vol}[r] = \frac{r^D \pi^{D/2}}{\Gamma[D/2+1]}, \tag{8.8}$$

where $\Gamma[\bullet]$ is the Gamma function. Show using Stirling's formula that the volume of a hypersphere of diameter one (radius $r=0.5$) becomes zero as the dimension increases.

Appendix B.1.3 Gamma function

Appendix B.1.4 Stirling's formula

Problem 8.8* Consider a hypersphere of radius $r = 1$. Find an expression for the proportion of the total volume that lies in the outermost 1% of the distance from the center (i.e., in the outermost shell of thickness 0.01). Show that this becomes one as the dimension increases.

Problem 8.9 Figure 8.13c shows the distribution of distances of samples of a standard normal distribution as the dimension increases. Empirically verify this finding by sampling from the standard normal distributions in 25, 100, and 500 dimensions and plotting a histogram of the distances from the center. What closed-form probability distribution describes these distances?