Measuring performance

Measuring performance

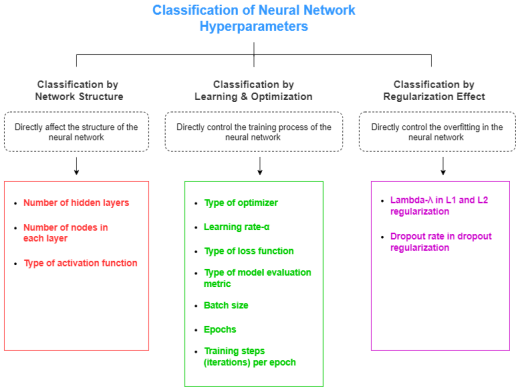
Steven H. Frankel

Technion

February 20, 2024

Outline

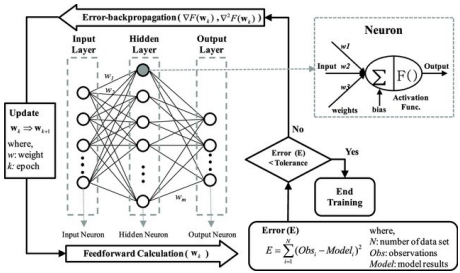
- Introduction
- Training simple model
- Sources of error
- Reducing error
- Double descent
- Choosing hyperparameters
- Summary

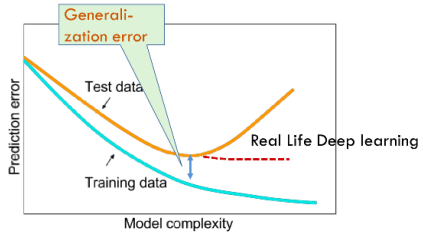


Understanding Deep Learning, Chapter 8

Introduction

- Previously, we described NNs, loss functions, and training algorithms
- Here, we consider *how to measure performance of trained models*
- With sufficient capacity (i.e. number of hidden units), a NN model will often perform perfectly on training data
- However, this does not necessarily mean it will *generalize* well to new *test data*





Introduction

- Test errors have three causes:
  1. inherent uncertainty in task
  2. amount of training data
  3. choice of model
- The last item brings the issue of *hyperparameter search*
- Hence, we will discuss no. hidden layers, no. hidden units in each, and learning algorithm hyperparameters (e.g. learning rate and batch size)

Training a simple model

- To explore model performance, we consider using MNIST-1D[1] dataset (Fig. 8.1)
- See also here
- Consists of ten classes $y \in \{0, 1, \ldots, 9\}$ representing digits $0 - 9$

[1] MNIST-1D, MNIST-1D in Google colab

MNIST Dataset - Full 2D

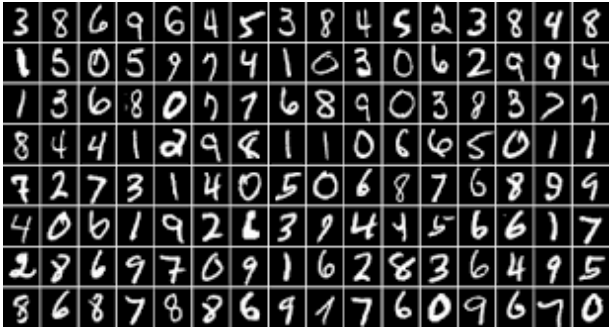


Figure: The MNIST database contains 60,000 training images and 10,000 testing images; each image 28×28 ($= 784$) grayscale pixels

Training a simple model

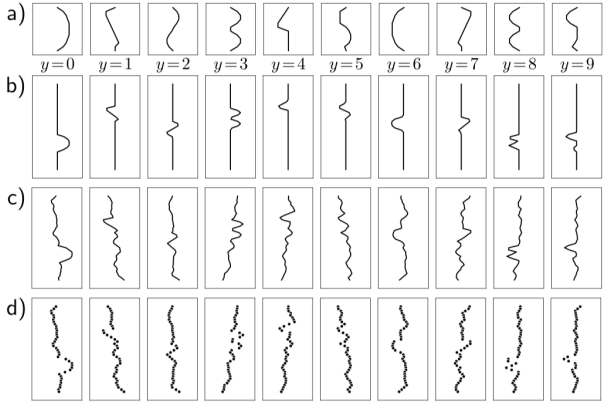


Figure 8.1 MNIST-1D. a) Templates for 10 classes $y \in \{0, \ldots, 9\}$, based on digits 0–9. b) Training examples $\mathbf{x}$ are created by randomly transforming a template and c) adding noise. d) The horizontal offset of the transformed template is then sampled at 40 vertical positions. Adapted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

Training a simple model

- Data *derived* from 1D templates for each of the digits
- Each data example $\mathbf{x}$ created by randomly transforming one of these templates and adding noise
- 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 ~ 400 examples of each class

Training a simple model

- We use network with $D_i = 40$ inputs and $D_0 = 10$ outputs which are passed through a *softmax function* to produce class probabilities
- Network has 2 hidden layers with $D = 100$ hidden units each
- It is trained using SGD with batch size 100 and learning rate 0.01 for 6000 steps (150 epochs) with *multi-class cross-entropy loss*
- Fig. 8.2 shows training error decrease as training proceeds
- Training data classified perfectly after about 4000 steps
- Training loss also decreases, eventually approaching zero
- But this does not mean classifier is perfect!

Training a simple model

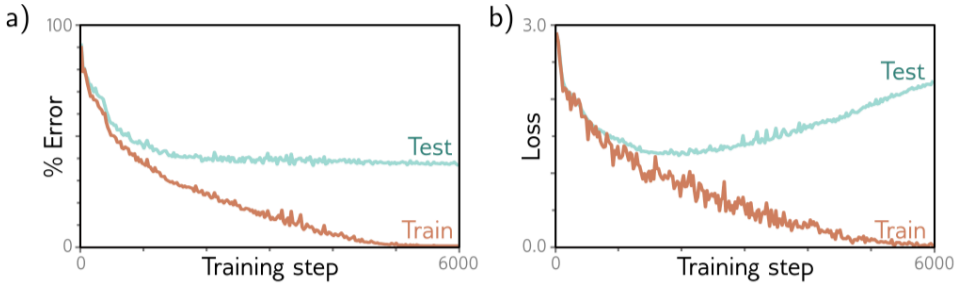


Figure 8.2 MNIST-1D results. a) Percent classification error as a function of the training step. The training set errors decrease to zero, but the test errors do not drop below $\sim 40\%$. This model doesn't generalize well to new test data. b) Loss as a function of the training step. The training loss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

Training a simple model

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

Training a simple model

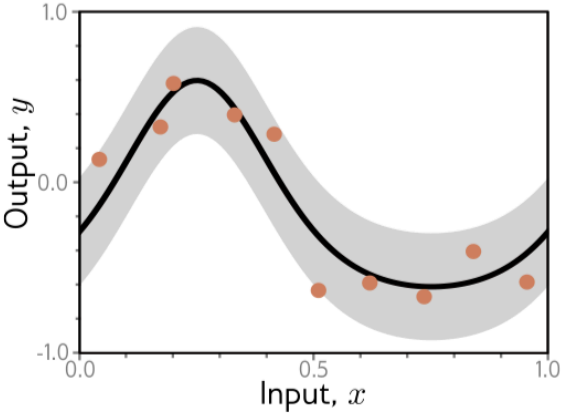
- Test loss (Fig. 8.2b) decreases for first 1500 training steps but then increases again
- At this point, test error fairly constant
- The model makes some mistakes but with increasing confidence (it *thinks* there are no more changes to make)
- This decreases probability of the correct answers and thus increases the negative log-likelihood
- This increasing confidence is a side-effect of softmax function
- Pre-softmax activations are driven to increasingly extreme values to make probability of training data approach one

Sources of error

- Let's consider sources of error when a model fails to generalize
- To visualize, we go back to 1D *linear least squares regression problem* where we know how ground truth data were generated
- Fig. 8.3 shows *quasi-sinusoidal function*
- Both training and test data generated by sampling input values in range $[0, 1]$
- Then passing them through this function
- Adding Gaussian noise with fixed variance

Sources of error

Figure 8.3 Regression function. Solid black line shows the 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.



Sources of error

- We fit simplified shallow NN to this data (Fig. 8.4)
- Weights/biases that connect input to hidden layer chosen so that the "joints" of the function are evenly spaced across 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 range $[0, 1]$
- Model is easy to understand and can be fit in closed form without need for stochastic optimization
- So we can guarantee to find global minimum of loss function during training

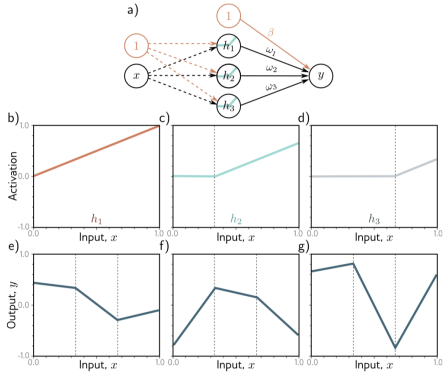


Figure 8.4 Simplified neural network with three hidden units. a) The weights and biases between the input and hidden layer are fixed (dashed arrows). b–d) They are chosen so that the hidden unit activations have slope one, and their joints are equally spaced across the interval, with joints at $x = 0$, $x = 1/3$, 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 $\boldsymbol{\phi}$.

Noise, bias, and variance

- Three possible sources of error: *noise, bias*, and *variance*, resp. (Fig. 8.5)
  - Noise: Data generation process includes addition of noise e.g. multiple possible valid outputs y for each input x (Fig. 8.5a); This error source is insurmountable for test data; training performance can still be perfect
  - Bias: Model not flexible enough to fit true function perfectly e.g. 3 region NN cannot model exactly quasi-sinusoidal function even when parameters chosen optimally (Fig. 8.5b)
  - Variance: With limited training examples, can't distinguish systematic changes in underlying function from noise in underlying data (Fig. 8.5c); this variability in fitted function is called variance

Training a simple model

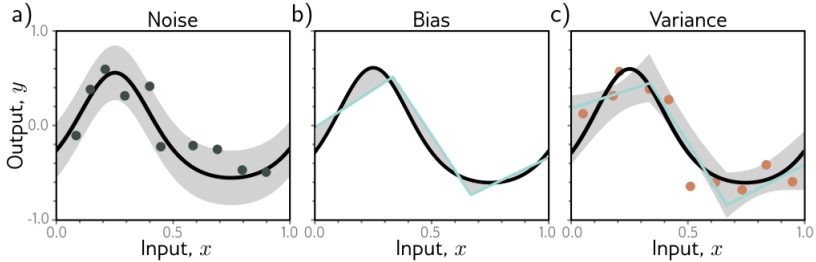


Figure 8.5 Sources of test error. a) Noise. Data generation is noisy, so even if the model exactly replicates the true underlying function (black line), the noise in the test data (gray points) means that some error will remain (gray region represents two standard deviations). b) Bias. Even with the 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.

BIAS-VARIANCE TRADEOFF

The bias-variance tradeoff represents the balance between a model's simplicity (bias) and its sensitivity to fluctuations in the training data (variance). Striking the right balance minimizes the total error and achieves the best performance.

Bias² is the squared difference between the model's average prediction and the true values

Irreducible Error is the inherent noise in the data, which cannot be reduced by improving the model

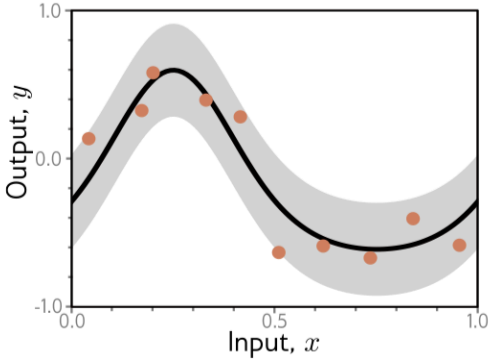
$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Variance measures the model's inconsistency across different training sets

Mathematical formulation of test error

- Consider 1D regression problem where data generation process has additive noise with variance σ^2 (see Fig. 8.3)

Figure 8.3 Regression function. Solid black line shows the 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.



Mathematical formulation of test error

- We can observe different outputs y for same input x, so for each x, there is a distribution $Pr(y|x)$ with expected value (mean) $\mu[x]$:

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

and fixed noise $\sigma^2 = \mathbb{E}_y[(\mu[x] - y[x])^2]$

Mathematical formulation of test error

- Now consider least squares loss between model prediction $f[x, \phi]$ at position x and observed value $y[x]$ at that position:

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

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

Mathematical formulation of test error

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

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

- Expected loss broken into two terms:
  1. squared deviation between model and true function mean
  2. second term is the noise

Mathematical formulation of test error

- The first term can be further partitioned into *bias* and *variance*
- The parameters ϕ of model $f[x, \phi]$ depend on training dataset $\mathcal{D} = \{x_i, y_i\}$ so we should write $f[x, \phi(\mathcal{D})]$
- Training dataset is a random sample from data generation process
- With a different sample of training data, we would learn different parameter values
- The expected model output $f_\mu[x]$ wrt all possible datasets $\mathcal{D}$ is:

$$f_\mu[x] = \mathbb{E}_\mathcal{D}[f[x, \phi(\mathcal{D})]]$$

- Can show, for given dataset $\mathcal{D} = \{x_i, y_i\}$:

$$\mathbb{E}_\mathcal{D}\Big[\mathbb{E}_y[L[x]]\Big] = \underbrace{\mathbb{E}_\mathcal{D}\Big[(f[x, \phi[\mathcal{D}]] - f_\mu[x])^2\Big]}_{\textit{variance}} + \underbrace{(f_\mu[x] - \mu[x])^2}_{\textit{bias}} + \underbrace{\sigma^2}_{\textit{noise}}$$

Reducing variance

$$\mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_{y}[L[x]]\Big] = \underbrace{\mathbb{E}_{\mathcal{D}}\Big[(f[x,\phi[\mathcal{D}]] - f_{\mu}[x])^2\Big]}_{variance} + \underbrace{(f_{\mu}[x] - \mu[x])^2}_{bias} + \underbrace{\sigma^2}_{noise}$$

- Equation says expected loss after considering uncertainty in training data $\mathcal{D}$ and test data y consists of 3 additive components
- *Variance* results from limited noisy training data
- Fitting model to two different training sets results in slightly different parameters
- So we can reduce variance by increasing quantity of training data
- This averages out inherent noise and ensures input space is well sampled
- Fig. 8.6 shows effect of training with $6, 10$ and 100 samples
- As we increase no. samples, fitted models become very similar and variances reduces
- *Adding training data almost always improves test performance*

Reducing variance

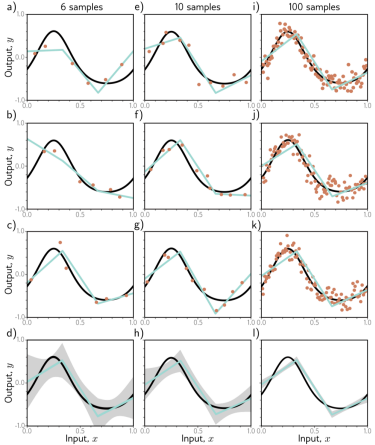


Figure 8.6 Reducing variance by increasing training data. a–c) The three-region model fitted to three different randomly sampled datasets of six points. The fitted model is quite different each time. d) We repeat this experiment many times and plot the mean model predictions (cyan line) and the variance 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.

Reducing bias

- *Bias* results from inability of model to describe true underlying function
- How to reduce this error? Making model more flexible e.g. increasing model capacity
- For NNs, this means adding more hidden units and/or hidden layers
- In simplified model, adding more hidden units so interval $[0, 1]$ is divided into more linear regions
- Fig. 8.7c show this *reduces bias* as we increase number of linear regions to ten
- Model becomes flexible enough to fit true function closely

Reducing bias

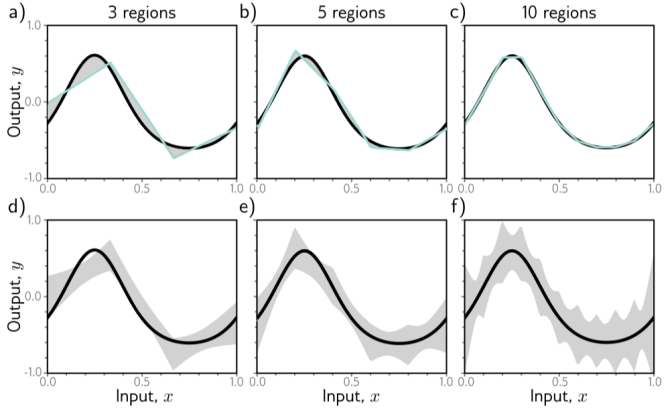


Figure 8.7 Bias and variance as a function of model capacity. a–c) As we increase the number of hidden units of the toy model, the number of linear regions increases, and the model becomes able to fit the true function closely; the bias (gray region) decreases. d–f) Unfortunately, increasing the model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

Bias-variance tradeoff

- Figs. 8.7d-f shown *unexpected side-effect of increasing model capacity*
- For fixed-size training dataset, *variance term increases as model capacity increases*
- *Consequently, increasing model capacity does not necessarily reduce test error*
- This is called *bias-variance trade-off*

Bias-variance tradeoff

- In Fig. 8.8, panels a-c, we fit simplified three-region model to three different datasets of fifteen points
- Although datasets differ, final model is much the same
- The noise in dataset roughly averages out in each linear region
- In panels d-f, we fit model with ten regions to same three datasets
- Model has more flexibility and will fit training data better
- But that extra power going to modeling noise e.g. *overfitting*

Bias-variance tradeoff

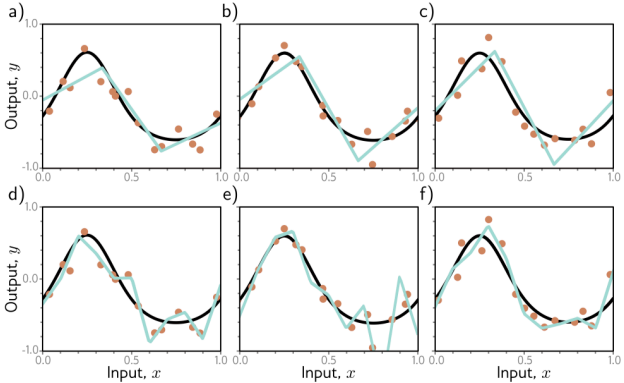


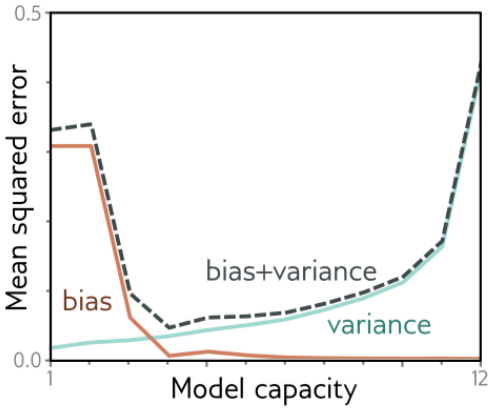
Figure 8.8 Overfitting. a–c) A model with three regions is fit to three different datasets of fifteen points each. The result is similar in all three cases (i.e., the variance is low). d–f) A model with ten regions is fit to the same datasets. The additional flexibility does not necessarily produce better 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.

Bias-variance tradeoff

- Adding capacity to model, decreases bias but increases variance for fixed-size training dataset
- This suggests there is optimal capacity where bias not too large and variance still relatively small
- Fig. 8.9 shows this
- For regression model, total expected error is sum of bias and variance
- This sum is minimized when model capacity is four (four hidden units and four linear regions)

Bias-variance tradeoff

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 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.



Coffee break

Double descent

- Return to MNIST-1D dataset
- Now we use $10,000$ training examples, test with another 5000 examples
- Examine training and test performance as we increase capacity (no. parameters) in model
- We train with Adam and step size 0.005 using full batch of $10,000$ examples for 4000 steps
- Fig. 8.10a shows training and test error for NN with 2 hidden layers as number of hidden units increases

Double descent

- Training error decreases as capacity grows and quickly becomes close to zero
- Vertical dashed line represents capacity where model has same no. parameters as there are training examples
- But model memorizes dataset before this point
- The test error increases as we add model capacity but does not increase as predicted by bias-variance trade-off curve; it keeps decreasing

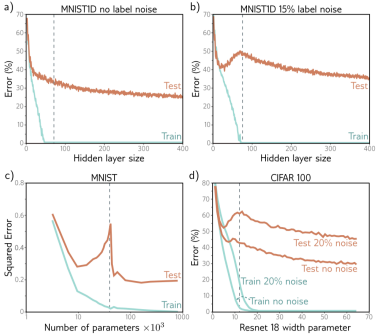


Figure 8.10 Double descent. a) Training and test loss on MNIST-1D for a two-hidden layer network as we increase the number of hidden units (and hence parameters) in each layer. The training loss decreases to zero as the number of parameters approaches the number of training examples (vertical dashed 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.

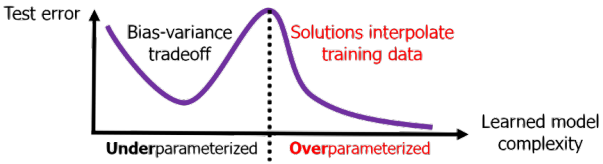
Double descent

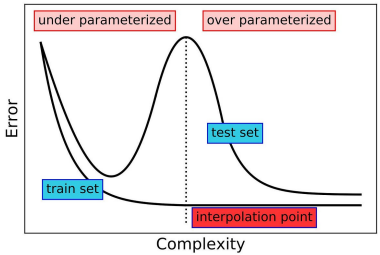
- In Fig. 8.10b, we repeat experiment but this time we randomize 15% of training labels
- Again, training error decreases to zero
- This time, there is more randomness, and the model requires almost as many parameters as there are data points to memorize the data
- The test error does show the typical 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

Double descent

- This is called *double descent*
- For some datasets like MNIST, it is present with original data (Fig. 8.10c)
- For others, like MNIST-1D and CIFAR (Fig. 8.10d), it emerges when adding noise to labels
- First part of curve called *classical* or *under-parameterized regime*
- Second part as *modern* or *over-parameterized regime*
- Central part where error increases is called *critical regime*

Double descent concept





Double descent - Explanation

- Due to two phenomena
- First, test performance becomes temporarily worse when model has just enough capacity to memorize data
- Second, test performance continues to improve with capacity even after training performance is perfect
- The first is related to bias-variance tradeoff
- The second is more confusing e.g. why performance better in over-parameterized regime?
- See text for more discussion

Double descent - Explanation

- To understand this, consider that once model has enough capacity to drive training loss to near zero, the model fits the training data almost perfectly
- This implies further capacity cannot help the model fit the

Increasing capacity

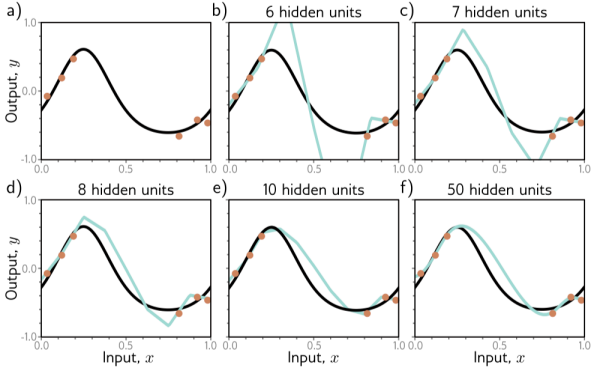


Figure 8.11 Increasing capacity (hidden units) allows smoother interpolation between sparse data points. a) Consider this situation where the training data (orange circles) are sparse; there is a large region in the center with no data examples to constrain the model to mimic the true function (black 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.

Regularization

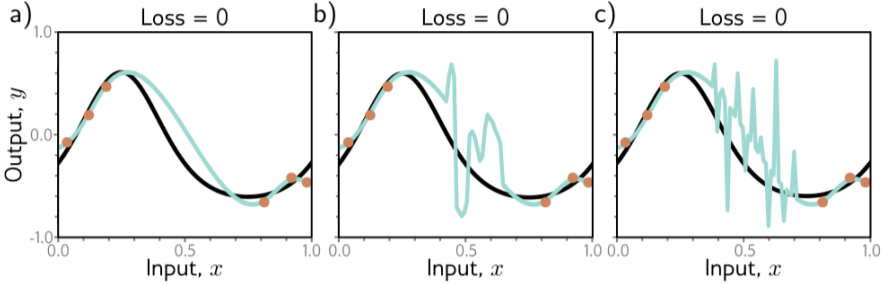


Figure 8.12 Regularization. a–c) Each of the three fitted curves passes through the data points exactly, so the training loss for each is zero. However, we might expect the smooth curve in panel (a) to generalize much better to new data than the erratic curves in panels (b) and (c). Any factor that biases 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.

Choosing hyperparameters

- *Hyperparameter search* includes finding best hyperparameters e.g. no. hidden layers/units per layer, learning rate, etc.
- Empirically chosen; validation data set (third set)
- See text for more discussion

Summary

- To measure performance, we use separate test set e.g. generalization
- Errors due to noise, bias, and variance
- Adding training data decreases variance
- When model capacity less than no. training examples, increasing capacity decreases bias but increases variance
- This is bias-variance tradeoff e.g. there is a capacity where trade-off is optimal
- Tendency for performance to improve with capacity, even when parameters exceed training examples create double descent curve

Double descent

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

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

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

March 2023

Abstract

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

1 What is double descent?

Double descent

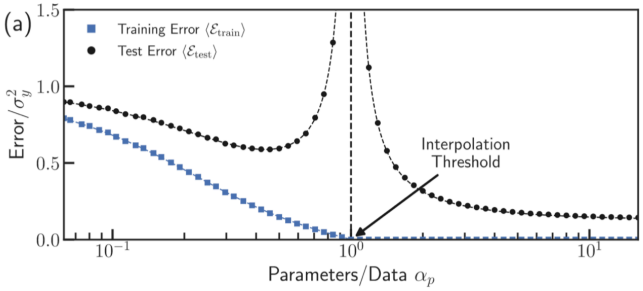


Figure 1: Double descent: test error falls, rises, then falls as a ratio of parameters to data. Fig. 3A from [18].

Regularization

Regularization

Steven H. Frankel

Technion

February 20, 2024

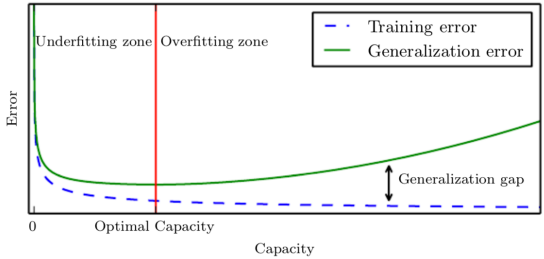
Outline

- Previously we discussed how to measure model performance
- We noted a potential *performance gap* between training and test data
- This could be due to overfitting or due to model being unconstrained in areas with no training examples
- Here, we address this using *regularization* techniques

Understanding Deep Learning, Chapter 9

Introduction

- Regularization is family of methods to reduce this generalization gap



- It involves *adding explicit terms to loss function* that favor certain parameter choices
- We start showing how SGD results in *implicit regularization*
- Then consider *early stopping*, *ensembling*, *dropout*, *label smoothing*, and *transfer learning*

Explicit regularization

- Consider fitting a model $f[\mathbf{x}, \boldsymbol{\phi}]$ with parameters $\boldsymbol{\phi}$ using training set $\{\mathbf{x}_i, \mathbf{y}_i\}$ of input/output pairs
- We seek minimum of loss function $L[\boldsymbol{\phi}]$:

$$\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}}[L[\boldsymbol{\phi}]] = \arg\min_{\boldsymbol{\phi}}\left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i]\right]$$

- To bias this minimization toward certain solutions, we include additional term:

$$\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}}\left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i] + \lambda \cdot g[\boldsymbol{\phi}]\right]$$

where $g[\boldsymbol{\phi}]$ is function that returns a scalar that takes a larger value when the parameters are less preferred

Explicit regularization

- The term λ is positive scalar that controls the relative contribution of original loss function and regularization term
- The minima of the regularized loss function usually differ from those in the original, so the training procedure converges to different parameter values (see Fig. 9.1)

Explicit regularization

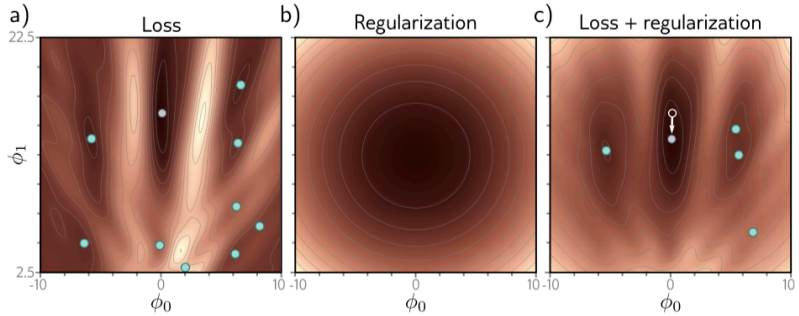


Figure 9.1 Explicit regularization. a) Loss function for Gabor model (see section 6.1.2). Cyan circles represent local minima. Gray circle represents the global minimum. b) The regularization term favors parameters close to the center of the plot by adding an increasing penalty as we move away from this point. c) The final loss function is the sum of the original loss function plus the regularization term. This surface has fewer local minima, and the global minimum has moved to a different position (arrow shows change).

Regularization from a probabilistic perspective

- Recall in Section 5.1, we showed how loss functions are constructed from the *maximum likelihood criterion*:

$$\hat{\phi} = \arg\max_{\phi} \Big[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i, \phi) \Big]$$

- *Regularization* can be seen as a *prior* $Pr(\phi)$ that represents knowledge about the parameters before we observe the data:

$$\hat{\phi} = \arg\max_{\phi} \Big[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i, \phi) Pr(\phi) \Big]$$

- Moving back to negative log-likelihood loss function by taking log and multiplying by minus one, we see $\lambda \cdot g[\phi] = -\log[Pr(\phi)]$

$L2$ regularization

- Most commonly used regularization term is $L2$ *norm*, which penalizes sum of squares of the parameter values themselves:

$$\hat{\phi} = \arg\min_{\phi} \left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i] + \lambda \sum_{j} \phi_j^2 \right]$$

  where j indexes the parameters
- For NNs, L2 regularization is usually applied to the weights but not the biases
- Hence it is called *weight decay term*
- This encourages smaller weights, so output function is smoother
- See text for more discussion

L2 regularization

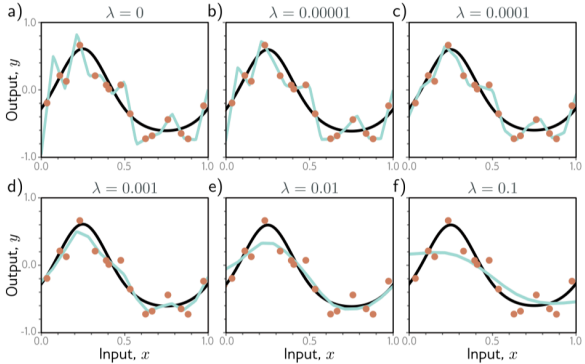


Figure 9.2 L2 regularization in simplified network (see figure 8.4). a–f) Fitted functions as we increase the regularization coefficient λ. The black curve is the true function, the orange circles are the noisy training data, and the cyan curve is the fitted model. For small λ (panels a–b), the fitted function passes exactly through the data points. For intermediate λ (panels c–d), the function is smoother and more similar to the ground truth. For large λ (panels e–f), the fitted function is smoother than the ground truth, so the fit is worse.

Implicit regularization

- A recent finding shows that neither GD or SGD moves neutrally to minimum of loss function
- Each exhibits a preference for some solutions over others
- This is called *implicit regularization*

Implicit regularization in GD

- Consider a continuous version of GD where step size is infinitesimal
- Change of parameters will be governed by:

$$\frac{\partial \boldsymbol{\phi}}{\partial t} = -\frac{\partial L}{\partial \boldsymbol{\phi}}$$

- GD approximates this with series of discrete steps of size α:

$$\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}_t - \alpha \frac{\partial L[\boldsymbol{\phi}]}{\partial \boldsymbol{\phi}}$$

this results in deviation from continuous path (Fig. 9.3)

Implicit regularization in GD

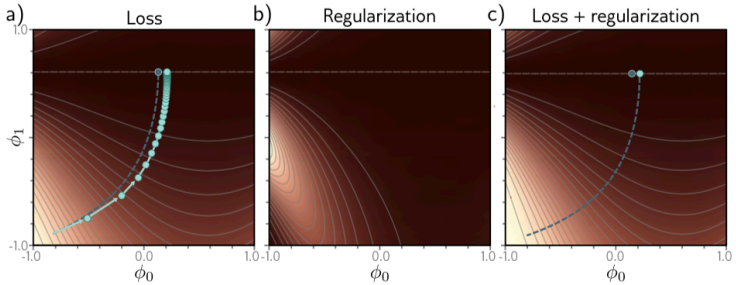


Figure 9.3 Implicit regularization in gradient descent. a) Loss function with family of global minima on horizontal line $\phi_1 = 0.61$. Dashed blue line shows continuous gradient descent path starting in bottom-left. Cyan trajectory shows discrete gradient descent with step size 0.1 (first few steps shown explicitly as arrows). The finite step size causes the paths to diverge and reach a different final position. b) This disparity can be approximated by adding a regularization term to the continuous gradient descent loss function that penalizes the squared gradient magnitude. c) After adding this term, the continuous gradient descent path converges to the same place that the discrete one did on the original function.

Implicit regularization in SGD

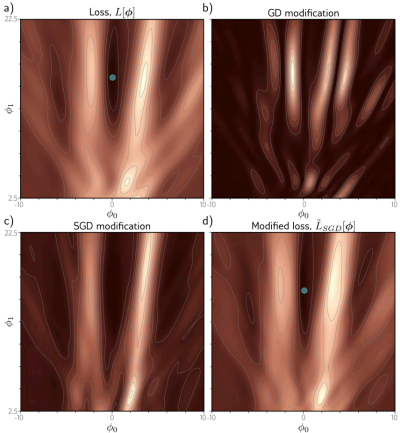


Figure 9.4 Implicit regularization for stochastic gradient descent. a) Original loss function for Gabor model (section 6.1.2). b) Implicit regularization term from gradient descent penalizes the squared gradient magnitude. c) Additional implicit regularization from stochastic gradient descent penalizes the variance of the batch gradients. d) Modified loss function (sum of original loss plus two implicit regularization components).

Effect of learning rate/batch size

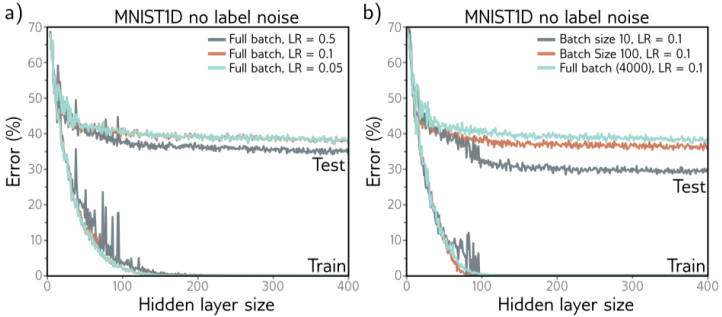


Figure 9.5 Effect of learning rate and batch size for 4000 training and 4000 test examples from MNIST-1D (see figure 8.1) for a neural network with two hidden layers. a) Performance is better for large learning rates than for intermediate or small ones. In each case, the number of iterations is 6000× the learning rate, so each solution has the opportunity to move the same distance. b) Performance is superior for smaller batch sizes. In each case, the number of iterations was chosen so that the training data were memorized at roughly the same model capacity.

Improving performance

- *Early stopping* - stopping training before full convergence e.g. reduces overfitting
- *Ensembling* - build several models and average their predictions; bootstrapping or bagging
- *Dropout* - randomly clamps a subset (50%) of hidden units to zero at each iteration of SGD
- *Applying noise* - e.g. to input data; weights; perturb labels
- *Bayesian inference*
- *Transfer learning and multi-task learning* - network is pre-trained to perform a related secondary task for which data are more plentiful; model then adapted to original task

Early stopping

- This refers to stopping training procedure before it fully converged
- This can reduce overfitting assuming model already capture coarse shape of underlying function but not had time to overfit the noise (Fig. 9.6)
- Since weights initialized to small values (Section 7.5), they don't have time to become large
- This is similar effect to explicit L2 regularization
- Early stopping features single hyperparameter: number of steps after which learning is terminated

Early stopping

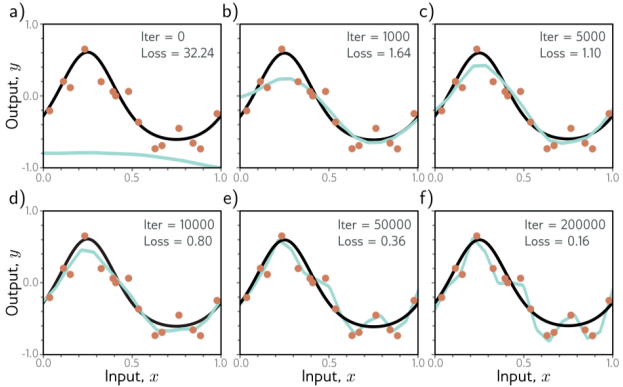


Figure 9.6 Early stopping. a) Simplified shallow network model with 14 linear regions (figure 8.4) is initialized randomly (cyan curve) and trained with SGD using a batch size of five and a learning rate of 0.05. b–d) As training proceeds, the function first captures the coarse structure of the true function (black curve) before e–f) overfitting to the noisy training data (orange points). Although the training loss continues to decrease throughout this process, the learned models in panels (c) and (d) are closest to the true underlying function. They will generalize better on average to test data than those in panels (e) or (f).

Ensemble methods

- Another approach to reducing generalization gap is build several models (*ensemble*) and average predictions
- Can use differential random initializations for each model
- Or re-sampling the training data with replacement and training a different model from each
- This is called *bootstrap aggregating* or *bagging* (Fig. 9.7)

Ensemble methods

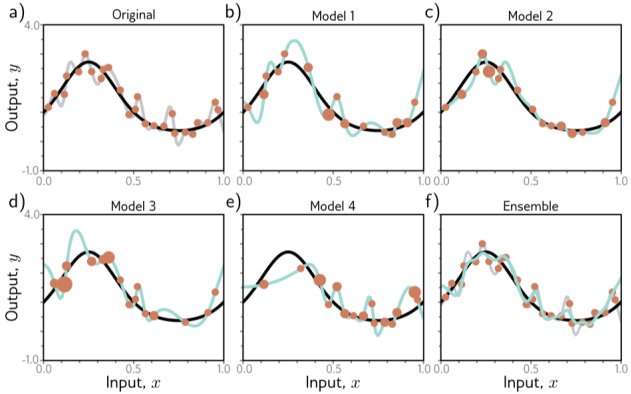


Figure 9.7 Ensemble methods. a) Fitting a single model (gray curve) to the entire dataset (orange points). b–e) Four models created by re-sampling the data with replacement (bagging) four times (size of orange point indicates number of times the data point was re-sampled). f) When we average the predictions of this ensemble, the result (cyan curve) is smoother than the result from panel (a) for the full dataset (gray curve) and will probably generalize better.

Dropout

- Dropout randomly clamps a subset (typically 50%) of hidden units to zero at each iteration of SGD (Fig. 9.8)
- Makes network less dependent on any given hidden unit
- This encourages weights to have smaller magnitudes so function less sensitive to presence of absence of the hidden unit
- Can eliminate undesirable kinks in function that are far from training data and don't affect loss
- Consider 3 hidden units that become sequentially active as we move along curve (Fig. 9.9a)

Dropout

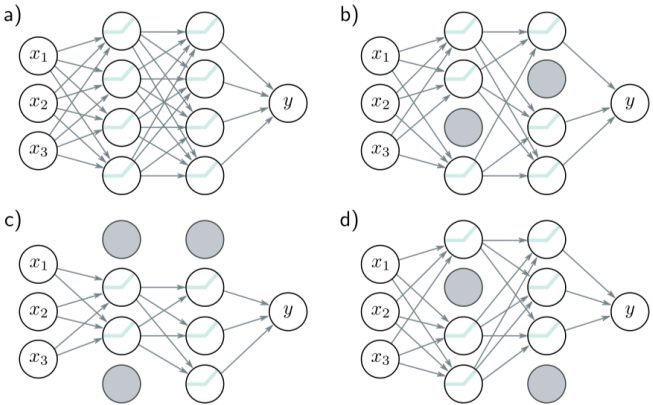


Figure 9.8 Dropout. a) Original network. b–d) At each training iteration, a random subset of hidden units is clamped to zero (gray nodes). The result is that the incoming and outgoing weights from these units have no effect, so we are training with a slightly different network each time.

Dropout

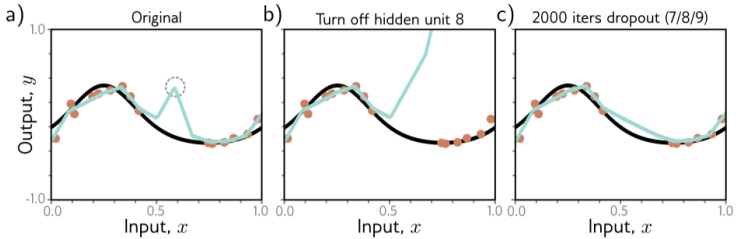


Figure 9.9 Dropout mechanism. a) An undesirable kink in the curve is caused by a sequential increase in the slope, decrease in the slope (at circled joint), and then another increase to return the curve to its original trajectory. Here we are using full-batch gradient descent, and the model already fits the data as well as possible, so further training won't remove the kink. b) Consider what happens if we remove the hidden unit that produced the circled joint in panel (a), as might happen using dropout. Without the decrease in the slope, the right-hand side of the function takes an upwards trajectory, and a subsequent gradient descent step will aim to compensate for this change. c) Curve after 2000 iterations of (i) randomly removing one of the three hidden units that cause the kink and (ii) performing a gradient descent step. The kink does not affect the loss but is nonetheless removed by this approximation of the dropout mechanism.

Adding noise to inputs

- Add noise to input data
- This smooths out the learned function
- For regression problems, its equivalent to adding regularizing term that penalizes derivatives of network's output wrt to its input
- An extreme version is *adversarial training*
- Here the optimization algorithm actively searches for small perturbations of the input that cause large changes to the output
- Another option is add noise to weights
- Finally, perturb labels

Adding noise to inputs

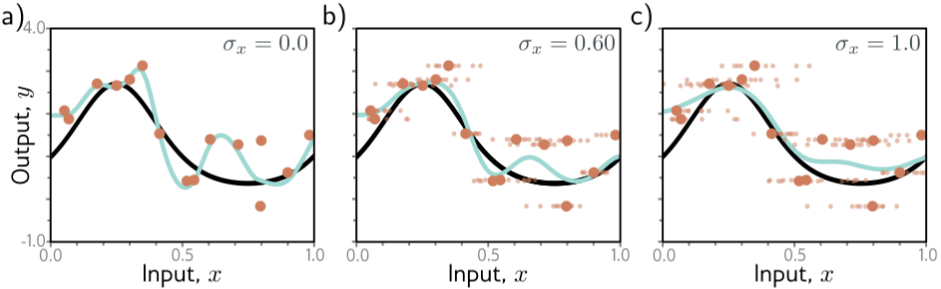


Figure 9.10 Adding noise to inputs. At each step of SGD, random noise with variance σ_x^2 is added to the batch data. a–c) Fitted model with different noise levels (small dots represent ten samples). Adding more noise smooths out the fitted function (cyan line).

Summary

- Explicit regularization involves adding extra term to loss function to change position of minimum
- SGD with finite step size has bias interpreted as implicit regularization
- Improve generalization by early stopping, dropout, ensembling, adding noise, transfer learning etc.

Summary

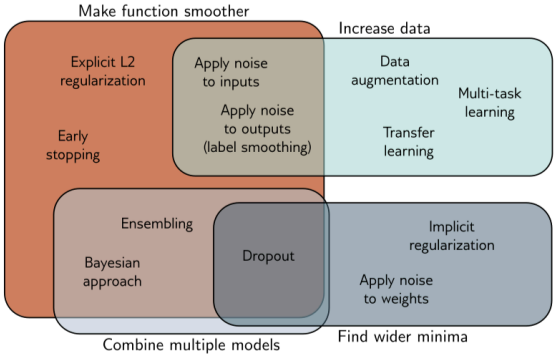


Figure 9.14 Regularization methods. The regularization methods discussed in this chapter aim to improve generalization by one of four mechanisms. Some methods aim to make the modeled function smoother. Other methods increase the effective amount of data. The third group of methods combine multiple models and hence mitigate against uncertainty in the fitting process. Finally, the fourth group of methods encourages the training process to converge to a wide minimum where small errors in the estimated parameters are less important (see also figure 20.11).