Deep Learning

Jasmine. Hao[1]

[1]University of Hong Kong

ECON 3225: Big Data Economics

Outline

Outline

What is a Neural Network?

- A neural network takes an input vector of p variables $X = (X_1, X_2, ..., X_p)$ and builds a nonlinear function $f(X)$ to predict the response Y.
- What distinguishes neural networks from previous methods (e.g., trees, boosting, and generalized additive models) is the particular **structure** of the model.

Neural Network Structure

- General Structure:
  - The neural network consists of interconnected layers of units, which process and transform the input data.
  - Each layer is composed of multiple units, also known as neurons or nodes, that perform computations.
- Input Layer:
  - The **input layer** contains the features $X_1, X_2, ..., X_p$, representing the input variables.
- Hidden Layer(s):
  - The **hidden layer(s)** lie between the input and output layers and are responsible for learning complex patterns and representations.
- Output Layer:
  - The **output layer** produces the final predictions or classifications.
- Connections:
  - Connections between units are represented by arrows, indicating the flow of information.

Example: Simple Feed-forward Neural Network

- Figure 10.1 shows a **simple feed-forward neural network** for modeling a quantitative response using $p = 4$ predictors.
- In this example, we have an input layer with four units representing the features $X_1, ..., X_4$.
- The arrows indicate that each input from the input layer is connected to each of the K hidden units.
- The number of hidden units, K, can be chosen based on the complexity of the problem.
- The hidden layer processes the input data, applying non-linear transformations and learning complex relationships.
- Finally, the output layer produces the predicted response.

[Illustration] Neural Network (Single Hidden Layer)

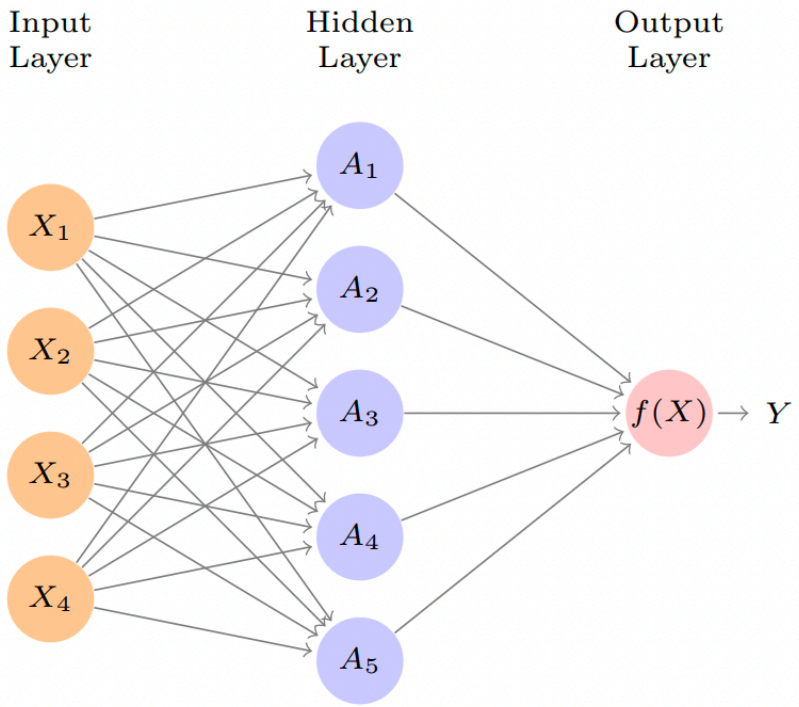


FIGURE 10.1. *Neural network with a single hidden layer. The hidden layer computes activations $A_k = h_k(X)$ that are nonlinear transformations of linear combinations of the inputs $X_1, X_2, \ldots, X_p$. Hence these A_k are not directly observed. The functions $h_k(\cdot)$ are not fixed in advance, but are learned during the training of the network. The output layer is a linear model that uses these activations A_k as inputs, resulting in a function $f(X)$.*

Neural Network Model

- The neural network model has the form:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j). \quad (10.1)$$

  which involves **activations** A_k computed from input features using a nonlinear **activation function** $g(z)$.

$$A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j), \quad (10.2)$$

- The activations from the hidden layer feed into the output layer, resulting in a linear regression model.

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k. \quad (10.3)$$

Activation Functions: Sigmoid vs. ReLU

Sigmoid In earlier instances of neural networks, the sigmoid activation function was favored. It is used in logistic regression to convert a linear function into probabilities.

$$g(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}, \tag{10.4}$$

ReLU (rectified linear unit) In modern neural networks, the ReLU activation function is preferred. It is more efficient to compute and store, and while it thresholds at zero, the constant term w_{k0} shifts this inflection point.

$$g(z) = (z)_+ = \begin{cases} 0 & z < 0 \\ z & otherwise \end{cases}, \tag{10.5}$$

[Illustration] Activation Functions

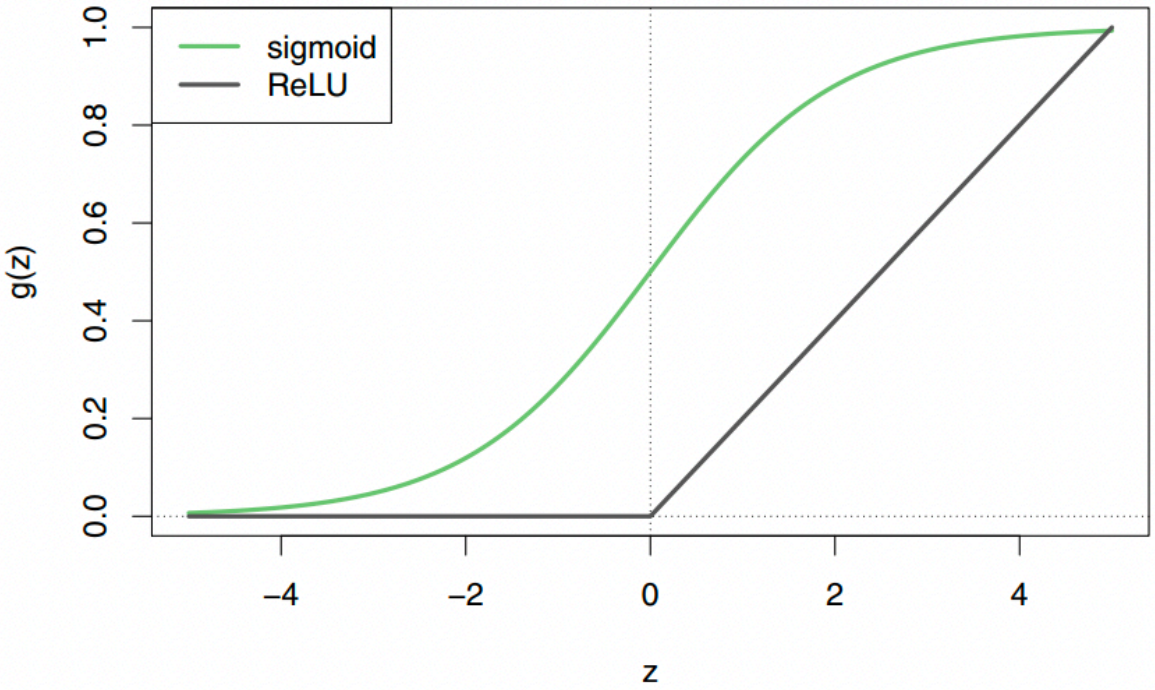


FIGURE 10.2. *Activation functions. The piecewise-linear* `ReLU` *function is popular for its efficiency and computability. We have scaled it down by a factor of five for ease of comparison.*

Nonlinearity in Activation Function: Importance

Nonlinearity: The nonlinearity in the activation function is essential for capturing complex nonlinearities and interaction effects in the model.

Nonlinearity in Activation Function: Example Parameters

Example: $p = 2$ input variables $X = (X_1, X_2)$ and $K = 2$ hidden units $h_1(X)$ and $h_2(X)$ with $g(z) = z^2$. Specify parameters as

$$\begin{array}{lll} \beta_0 = 0, & \beta_1 = \frac{1}{4}, & \beta_2 = -\frac{1}{4}, \\ w_{10} = 0, & w_{11} = 1, & w_{12} = 1, \\ w_{20} = 0, & w_{21} = 1, & w_{22} = -1. \end{array} \quad , \tag{10.6}$$

Nonlinearity in Activation Function: Hidden Units

From (10.2), this means that

$$\begin{aligned} h_1(X) &= (0 + X_1 + X_2)^2, \\ h_2(X) &= (0 + X_1 - X_2)^2. \end{aligned} \quad , \tag{10.7}$$

Then plugging (10.7) into (10.1), we get

$$f(X) = X_1 X_2. \tag{10.8}$$

Nonlinearity in Activation Function: Parameter Estimation

Parameter Estimation: Fitting a neural network requires estimating the unknown parameters in (10.1). For a quantitative response, typically squared-error loss is used, so that the parameters are chosen to minimize

$$\sum_{i=1}^{n}(y_i - f(x_i))^2. \tag{10.9}$$

Outline

Why Multilayer Neural Networks are Needed?

1. *Modern neural networks typically have more than one hidden layer and often many units per layer.*
    1. *In theory, a single hidden layer with a large number of units has the ability to approximate most functions.*
    2. However, the learning task of discovering a good solution is made much easier with multiple layers each of modest size.

Introduction to MNIST Handwritten Digit Dataset

- We will illustrate a large dense network on the famous and publicly available MNIST handwritten digit dataset[1].
- Figure 10.3 shows examples of these digits. There are 60,000 training images, and 10,000 test images.
- The goal is to build a model to classify the images into their correct digit class $0-9$. Every image has $p = 28 \times 28 = 784$ pixels, each an eight-bit grayscale value between 0 and 255.
- one-hot encoding: These pixels are stored in the input vector X. The output is the class label, represented by a vector $Y = (Y_0, Y_1, ..., Y_9)$ of 10 dummy variables, with a one in the position corresponding to the label, and zeros elsewhere.

[1] See LeCun, Cortes, and Burges (2010) "The MNIST database of handwritten digits", available at http://yann.lecun.com/exdb/mnist.

[Illustration] Examples from the MNIST Corpus

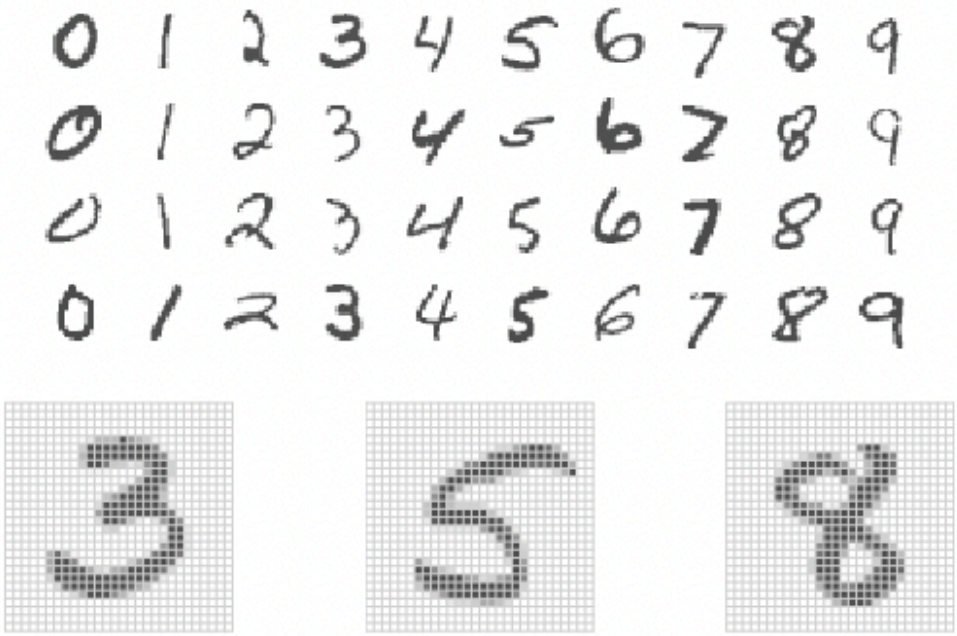


FIGURE 10.3. *Examples of handwritten digits from the* MNIST *corpus. Each grayscale image has* 28×28 *pixels, each of which is an eight-bit number (0–255) which represents how dark that pixel is. The first 3, 5, and 8 are enlarged to show their 784 individual pixel values.*

Historical Context and Significance of Neural Network Technology

- On a historical note, digit recognition problems were the catalyst that accelerated the development of neural network technology in the late 1980s at AT&T Bell Laboratories and elsewhere.
- Pattern recognition tasks of this kind are relatively simple for humans. Our visual system occupies a large fraction of our brains, and good recognition is an evolutionary force for survival.
- These tasks are not so simple for machines, and it has taken more than 30 years to refine the neural-network architectures to match human performance.

[Illustration] Neural Network (Two Hidden Layers)

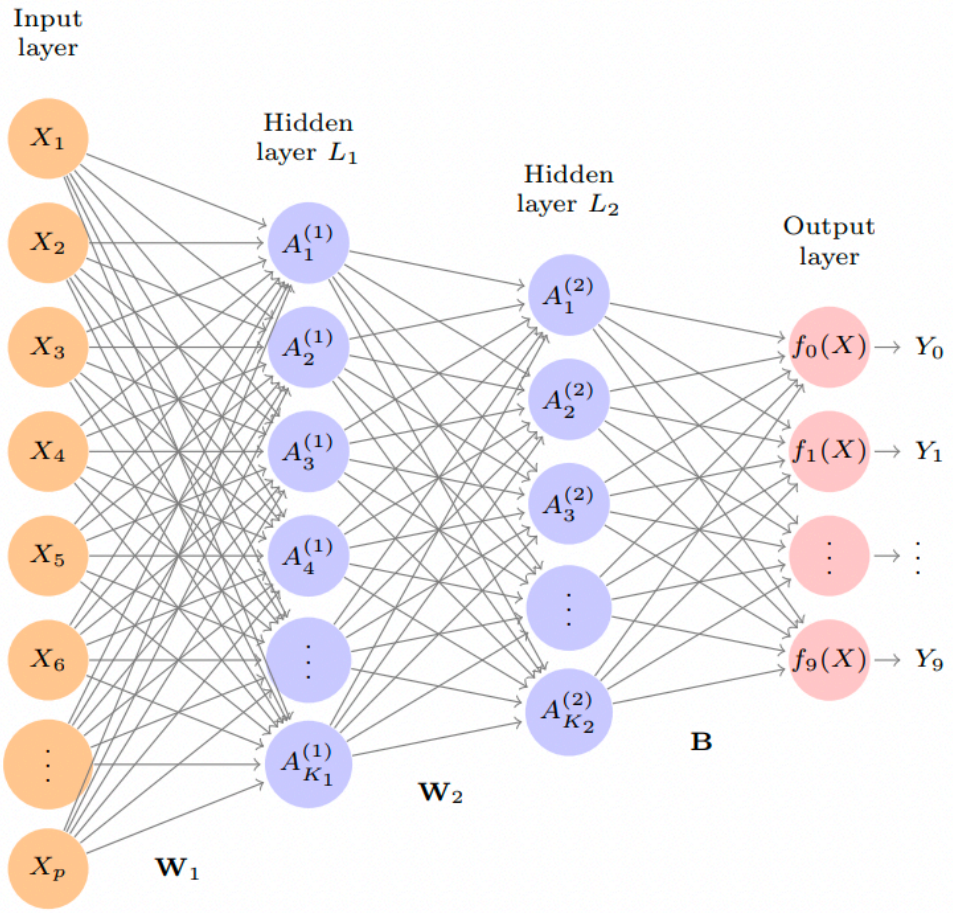


FIGURE 10.4. *Neural network diagram with two hidden layers and multiple outputs, suitable for the* MNIST *handwritten-digit problem. The input layer has* $p = 784$ *units, the two hidden layers* $K_1 = 256$ *and* $K_2 = 128$ *units respectively, and the output layer* 10 *units. Along with intercepts (referred to as* biases *in the deep-learning community) this network has 235,146 parameters (referred to as weights).*

Single Layer Networks vs Multilayer Networks

- Figure 10.4 shows a multilayer network architecture for digit-classification, with notable differences from Figure 10.1:
  - Two hidden layers L_1 (256 units) and L_2 (128 units), with an example of a network having seven hidden layers.
  - Ten output variables, representing a single qualitative variable (digit class 0–9), in contrast to one output variable.
  - In multi-task learning, a single network can predict multiple responses simultaneously, influencing the formation of the hidden layers.
  - The loss function is tailored for multiclass classification.

Compute New Activations: First Hidden Layer

- The first hidden layer is as in (10.2), with

$$A_k^{(1)} = hk^{(1)}(X) = f(X) = g(wk0^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j) \qquad (10.10)$$

for $k = 1, ..., K_1$.

Compute New Activations: Second Hidden Layer

- The second hidden layer treats the activations $A_k^{(1)}$ of the first hidden layer as inputs and computes new activations

$$A_l^{(2)} = hl^{(2)}(X) = f(X) = g(wl0^{(2)} + \sum_{k=1}^{K_1} w_{lk}^{(2)} A_k^{(1)}) \qquad (10.11)$$

for $l = 1, ..., K_2$. Notice that each of the activations in the second layer $A_l^{(2)} = hl^{(2)}(X)$ is a function of the input vector X.

Compute New Activations: Chain of Transformations

- This is the case because while they are explicitly a function of the activations $Ak^{(1)}$ from layer L_1, these in turn are functions of X.
- This would also be the case with more hidden layers.
- Thus, through a chain of transformations, the network is able to build up fairly complex transformations of X that ultimately feed into the output layer as features.

Weights and Bias

- We have introduced additional superscript notation such as $h_j^{(2)}(X)$ and $\omega_{lj}^{(2)}$ in (10.10) and (10.11) to indicate to which layer the activations and **weights** (coefficients) belong, in this case layer 2.
- The notation W_1 in Figure 10.4 represents the entire matrix of weights that feed from the input layer to the first hidden layer L_1.
  - This matrix will have 785×256 = 200,960 elements.
  - There are 785 rather than 784 because we must account for the intercept or **bias** term.[2]
- Each element $A_k^{(1)}$ feeds to the second hidden layer L_2 via the matrix of weights W_2 of dimension 257 × 128 = 32,896.

[2]The use of "weights" for coefficients and "bias" for the intercepts ω_{k0} in (10.2) is popular in the machine learning community; this use of bias is not to be confused with the "bias-variance" usage elsewhere in the book.

Output Layer

- The output layer consists of **10 responses** computed through linear models with weights stored in a matrix B.
- The **softmax** activation function is used to ensure that the 10 numbers behave like probabilities and sum to one.
- The classifier assigns the image to the class with the **highest probability**.
- To train the network, we minimize the **negative multinomial log-likelihood** (also known as the **cross-entropy**) to obtain the coefficient estimates.

Compare Test Performance

- Table 10.1 compares the test performance of the neural network with two simple models presented in Chapter 4 that make use of linear decision boundaries: multinomial logistic regression and linear discriminant analysis.
- The improvement of neural networks over both of these linear methods is dramatic: the network with dropout regularization achieves a test error rate below 2% on the 10,000 test images.

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

TABLE 10.1. *Test error rate on the* MNIST *data, for neural networks with two forms of regularization, as well as multinomial logistic regression and linear discriminant analysis. In this example, the extra complexity of the neural network leads to a marked improvement in test error.*

Overfitting and Regularization

- Adding the number of coefficients in W_1, W_2 and B, we get 235,146 in all, more than 33 times the number $785 \times 9 = 7{,}065$ needed for multinomial logistic regression.
  - Recall that there are 60,000 images in the training set. While this might seem like a large training set, there are almost four times as many coefficients in the neural network model as there are observations in the training set!
- To avoid overfitting, some regularization is needed. In this example, we used two forms of regularization:
  - **ridge regularization** (similar to ridge regression)
  - **dropout regularization**

Outline

Convolutional Neural Networks

- Neural networks had big successes in **image classification** around 2010, thanks to massive databases of labeled images.
- The CIFAR100 database consists of 60,000 images labeled according to **20 superclasses** with five classes per superclass.
- Each image has a resolution of 32×32 pixels with three eight-bit numbers per pixel representing **red, green, and blue**.
- The numbers for each image are organized in a three-dimensional array called a **feature map**.
- The first two axes are spatial, and the third is the **channel axis** representing the three colors.
- The database has a designated training set of 50,000 images and a test set of 10,000.

[Illustration] CIFAR100 Database

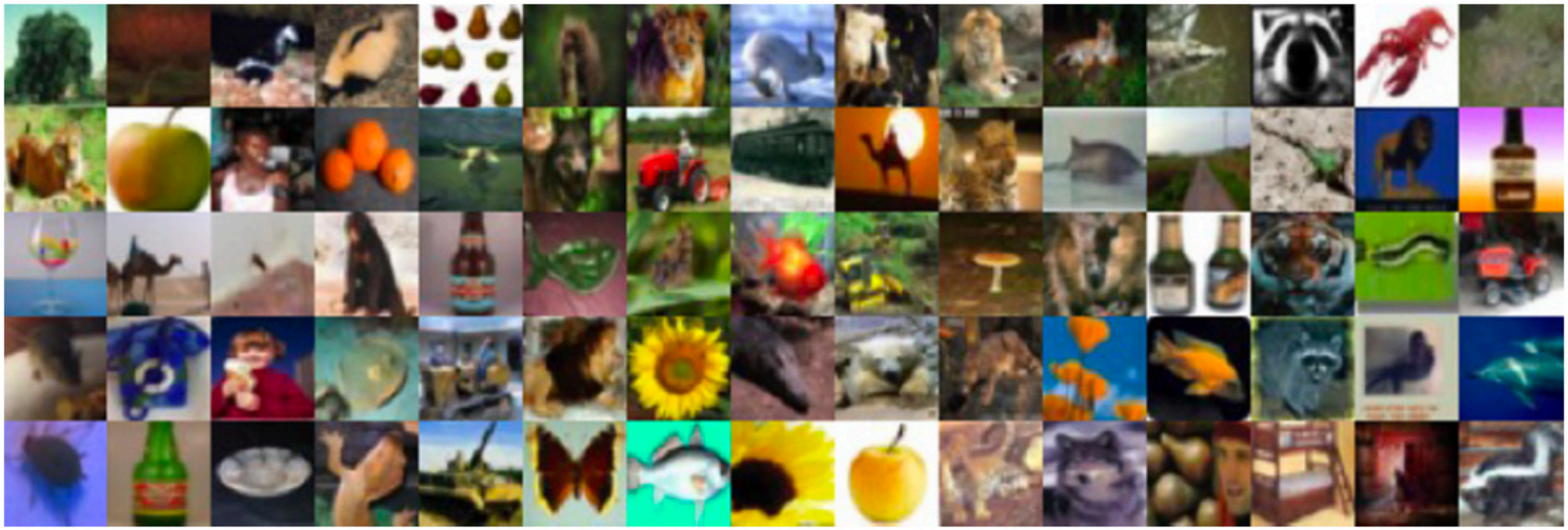


FIGURE 10.5. *A sample of images from the* `CIFAR100` *database: a collection of natural images from everyday life, with 100 different classes represented.*

CNNs for graphic classification

- A special family of **convolutional neural networks** (CNNs) has evolved for classifying images such as these, and has shown spectacular success on a wide range of problems.
- CNNs mimic to some degree how humans classify images, by recognizing specific features or patterns anywhere in the image that distinguish each particular object class. In this section we give a brief overview of how they work.
- Figure 10.6 illustrates the idea behind a convolutional neural network on a cartoon image of a tiger.[3]

[3] Thanks to Elena Tuzhilina for producing the diagram and https://www.cartooning4kids.com/ for permission to use the cartoon tiger.

[Illustration] An Example of CNN

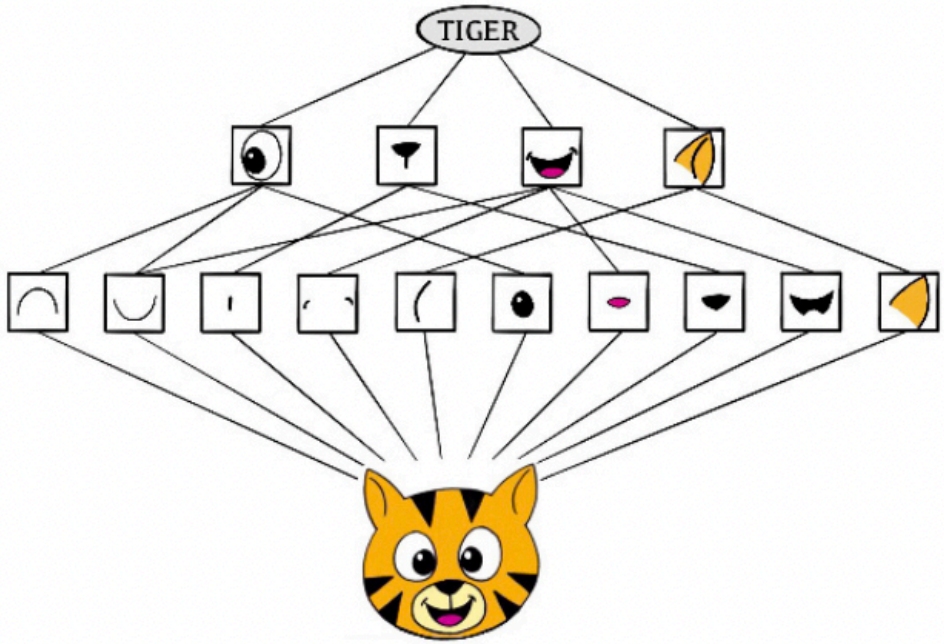


FIGURE 10.6. *Schematic showing how a convolutional neural network classifies an image of a tiger. The network takes in the image and identifies local features. It then combines the local features in order to create compound features, which in this example include eyes and ears. These compound features are used to output the label "tiger".*

How does CNN work?

- How does CNN work?
  1. The network first identifies low-level features in the input image, such as small edges, patches of color, and the like.
  2. These low-level features are then combined to form higher-level features, such as parts of ears, eyes, and so on.
  3. Eventually, the presence or absence of these higher-level features contributes to the probability of any given output class.
- How does a convolutional neural network build up this hierarchy? It combines two specialized types of hidden layers, called **convolution layers** and **pooling layers**.
  - Convolution layers search for instances of small patterns in the image.
  - Pooling layers downsample these to select a prominent subset.
  - In order to achieve state-of-the-art results, contemporary neuralnetwork architectures make use of many convolution and pooling layers.

Outline

Convolution Filter

- A **convolution layer** consists of multiple **convolution filters**, which are templates that detect local features in images.
- The convolution operation involves multiplying and adding matrix elements.
- To understand how a convolution filter works, consider an example:

$$\text{Original Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$$

$$\text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

  - When we convolve the image with the filter, we get the result

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$$

[Example] Convolution Filter

- Figure 10.7 illustrates the application of horizontal and vertical stripe filters to an image of a tiger.
  - A horizontal stripe filter highlights horizontal stripes and edges,
  - a vertical stripe filter emphasizes vertical stripes and edges.
- In a convolution layer, a bank of filters is used to pick out differently oriented edges and shapes in the image.

[Illustration] Vertical and Horizontal Stripes

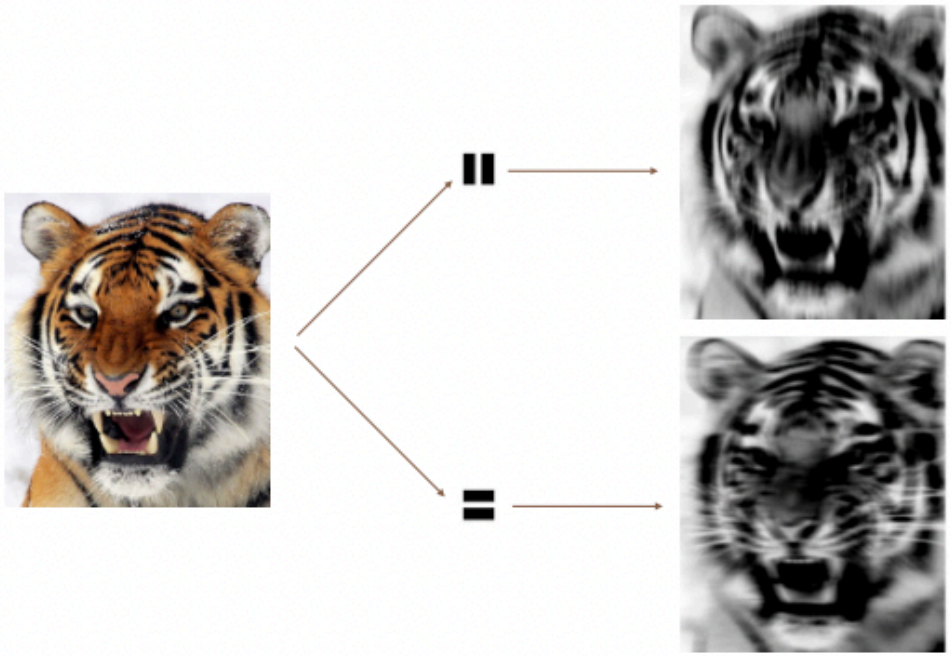


FIGURE 10.7. *Convolution filters find local features in an image, such as edges and small shapes. We begin with the image of the tiger shown on the left, and apply the two small convolution filters in the middle. The convolved images highlight areas in the original image where details similar to the filters are found. Specifically, the top convolved image highlights the tiger's vertical stripes, whereas the bottom convolved image highlights the tiger's horizontal stripes. We can think of the original image as the input layer in a convolutional neural network, and the convolved images as the units in the first hidden layer.*

Additional Details

- Color Images:
  - In a colour image, each channel **(red, green, blue)** is represented by a two-dimensional feature map, resulting in a three-dimensional feature map.
  - A convolution filter for a color image will also have three channels, with potentially different weights for each colour.
  - The results of the convolutions for each color channel are summed to form a two-dimensional output feature map.
- Multiple Convolution Filters:
  - Using K different convolution filters at the first hidden layer results in K two-dimensional output feature maps, treated as a three-dimensional feature map with K channels.
- Activation Function:
  - ReLU activation function (10.5) is typically applied to the convolved image, which can be viewed as a separate layer (detector layer) in the CNN.

Outline

Pooling Layers

- condenses a large image into a smaller summary image
  - a way to reduce image size and achieve location invariance.
- Max pooling is a common method where each non-overlapping 2 × 2 block of pixels is summarized using the maximum value in the block.
- Example of max pooling:

$$\text{Max Pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

Pooling

Figure: Mona Lisa in Pixels

Outline

Architecture of a CNN: Part 1

- A single convolution layer produces a new two-dimensional feature map:
  - the number of filters defining the channels in the resulting three-dimensional feature map.
- A pooling layer can be used to reduce the dimensions of the feature map.
- Figure 10.8 shows a typical architecture for a CNN, with multiple convolution and pooling layers.
  - Each convolution filter produces a new channel at the first hidden layer, resulting in a 32×32 feature map with more channels than the input color channels.
  - Max-pooling is used to reduce the size of the feature map in each channel by a factor of four.

Architecture of a CNN: Part 2

- This convolve-then-pool sequence is repeated for the next two layers.
  - Each subsequent convolution layer takes the previous layer's three-dimensional feature map as input and treats it as a single multi-channel image.
  - To compensate for the reduction in size of the feature map after each pooling layer, the number of filters in the next convolution layer is usually increased.
  - Sometimes multiple convolution layers are repeated before a pooling layer, effectively increasing the dimension of the filter.

[Illustration] Architecture of a Deep CNN

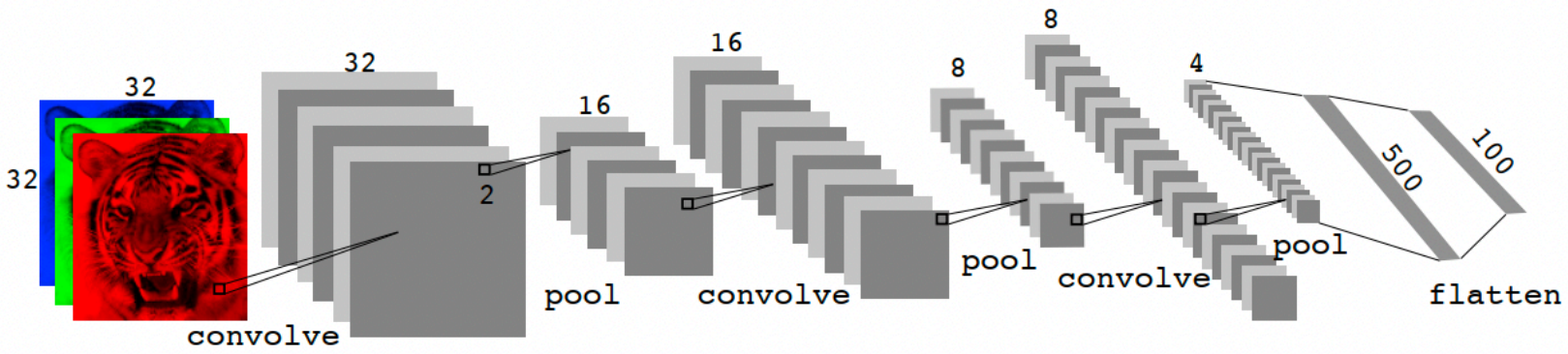


FIGURE 10.8. *Architecture of a deep CNN for the* `CIFAR100` *classification task. Convolution layers are interspersed with* 2×2 *max-pool layers, which reduce the size by a factor of 2 in both dimensions.*

Architecture of a CNN (cont'd)

- Convolution and Pooling:
  - The convolution and pooling operations are repeated until each channel feature map is reduced to just a few pixels in each dimension.
  - The resulting three-dimensional feature maps are flattened and fed into one or more fully-connected layers before reaching the output layer, which uses softmax activation for classification.
- Tuning Parameters:
  - Other tuning parameters include dropout learning and regularization techniques like lasso or ridge.
- Software Tools:
  - Software tools with extensive examples and guidance are available for constructing CNNs.
- Accuracy:
  - The best accuracy for the CIFAR100 official test set is just above 75%, with ongoing improvements likely.

Outline

Data Augmentation: An Introduction

- Data augmentation is a technique used in image modeling where each training image is replicated many times and randomly distorted in a natural way (see Figure 10.9).
- Typical distortions include zoom, shift, shear, rotation, and flips.

Data Augmentation: Benefits

- Data augmentation increases the training set with somewhat different examples, protecting against overfitting and acting as a form of regularization.

Data Augmentation: In Context of SGD

- Stochastic gradient descent algorithms for fitting deep learning models process randomly selected batches of training images, which works well with data augmentation as each image in the batch can be distorted on the fly (see Section 10.7.2).

[Illustration] Data Augmentation

FIGURE 10.9. *Data augmentation. The original image (leftmost) is distorted in natural ways to produce different images with the same class label. These distortions do not fool humans, and act as a form of regularization when fitting the CNN.*

Outline

[Illustration] Classification of Six Photographs

flamingo		Cooper's hawk		Cooper's hawk	
flamingo	0.83	kite	0.60	fountain	0.35
spoonbill	0.17	great grey owl	0.09	nail	0.12
white stork	0.00	robin	0.06	hook	0.07
Lhasa Apso		**cat**		**Cape weaver**	
Tibetan terrier	0.56	Old English sheepdog	0.82	jacamar	0.28
Lhasa	0.32	Shih-Tzu	0.04	macaw	0.12
cocker spaniel	0.03	Persian cat	0.04	robin	0.12

FIGURE 10.10. *Classification of six photographs using the* `resnet50` *CNN trained on the* `imagenet` *corpus. The table below the images displays the true (intended) label at the top of each panel, and the top three choices of the classifier (out of 100). The numbers are the estimated probabilities for each choice. (A kite is a raptor, but not a hawk.)*

Weight Freezing

- Pretrained hidden layers from models trained on massive corpora like imagenet can serve as features for general natural-image classification problems.
- Weight freezing can be used to train the last few layers of the network with much less data, after using pretrained hidden layers.
- The keras package and accompanying materials provide more details on weight freezing applications.

Predicting Attributes of Documents

- We introduce **predicting attributes of documents**, which has important applications in industry and science.
  - Examples of documents include articles in medical journals, news feeds, emails, tweets, etc.
  - Our example is IMDb ratings, where viewers have written critiques of movies, and the response is the **sentiment** of the review.

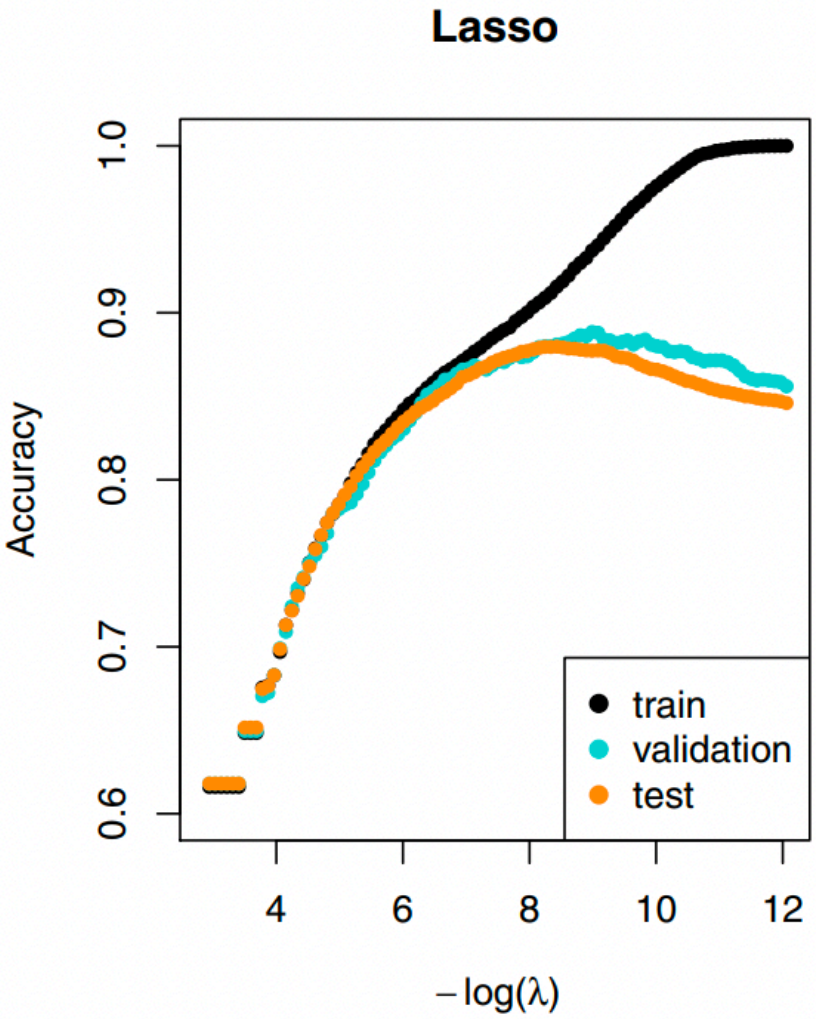
Bag-of-Words

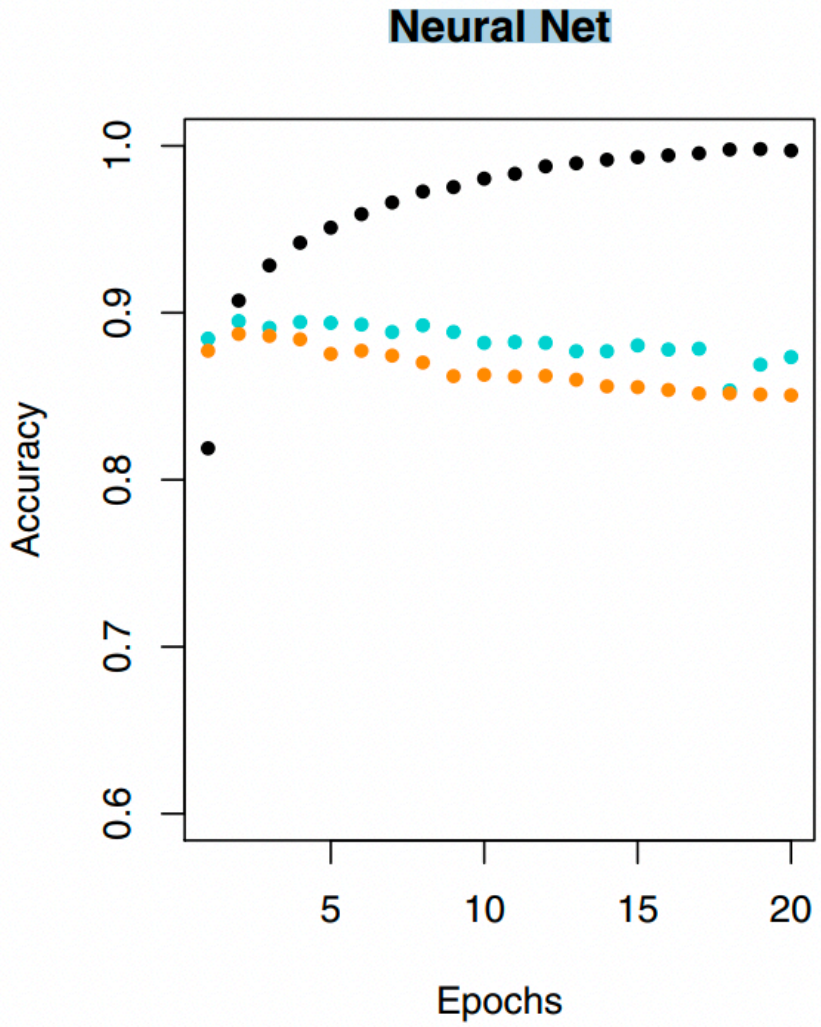
- To **featurize** a document, the simplest and most common method is the **bag-of-words** model.
  - Each document is scored for the presence or absence of each word in an English dictionary, creating a binary feature vector of length M (the number of words in the dictionary).
  - The dictionary is limited to the 10,000 most frequently occurring words in the training corpus of 25,000 reviews.
  - The resulting feature matrix is sparse, with mostly 0's and a few 1's in positions corresponding to words present in the document.
  - The training and test sets each have 25,000 examples, balanced with regard to sentiment.
  - The resulting training feature matrix X has dimension 25,000 × 10,000, but only 1.3
  - Such a matrix is stored efficiently in **sparse matrix format**.

Fitting Model Sequences: Handling Document Length

- Various methods can account for document length, such as recording the relative frequency of words instead of binary presence/absence.
- A validation set of size 2,000 is split from the 25,000 training observations for model tuning.
- Two model sequences are fit:
  - A lasso logistic regression using the glmnet package.
  - A two-class neural network with two hidden layers, each with 16 ReLU units.
- Both methods produce a sequence of solutions, indexed by the regularization parameter λ for lasso and number of gradient-descent iterations for neural network.
- Training accuracy increases monotonically in both cases, as shown in Figure 10.11.
- Validation error is used to select a good solution from each sequence (blue points in the plot) for making predictions on the test dataset.

[Illustration] LASSO and Neural Network





- Note that a two-class neural network amounts to a nonlinear logistic regression model. From (10.12) and (10.13) we can see that

$$log(\frac{Pr(Y=1|X)}{Pr(Y=0|X)}) = Z_1 - Z_0 = (\beta_{10} - \beta_{00}) + \sum_{l=1}^{K_2} (\beta_{1l} - \beta_{0l}) A_l^{(2)}. \quad (10.15)$$

  (This shows the redundancy in the softmax function; for K classes we really only need to estimate $K-1$ sets of coefficients. See Section 4.3.5.)
- In Figure 10.11 we show accuracy (fraction correct) rather than classification error (fraction incorrect), the former being more popular in the machine learning community.
- Both models achieve a test-set accuracy of about 88%.

Bag-of-N-Grams

- The bag-of-words model summarizes a document by the words present, and ignores their context.
- There are at least two popular ways to take the context into account:
  - The **bag-of-n-grams** model. For example, a bag of 2-grams records the consecutive co-occurrence of every distinct pair of words. “Bliss-fully long” can be seen as a positive phrase in a movie review, while “blissfully short” a negative.
  - Treat the document as a sequence, taking account of all the words in the context of those that preceded and those that follow.
- In the next section we explore models for sequences of data, which have applications in weather forecasting, speech recognition, language translation, and time-series prediction, to name a few. We continue with this IMDb example there.

Outline

Special Treatment for Sequential Data

- Many data sources are **sequential** and require special treatment when building predictive models, such as:
  - **Documents** like book and movie reviews, newspaper articles, and tweets, where the sequence and relative positions of words capture narrative, theme, and tone.
  - **Time series** of temperature, rainfall, financial market indices, etc., where we may want to forecast future values.
  - **Recorded speech** and musical recordings, where we may want to transcribe speech or assess the quality of music.
  - **Handwriting**, such as doctor's notes or handwritten digits, where we want to turn it into digital text or read digits using optical character recognition.

Input and Output of RNN

- In a **recurrent neural network** (RNN), the input object X is a sequence.
  - Consider a corpus of documents, such as the collection of IMDb movie reviews.
  - Each document can be represented as a sequence of L words, so $X = \{X_1, X_2, ..., X_L\}$,where each X_l represents a word.
  - The order of the words, and closeness of certain words in a sentence, convey semantic meaning.
  - RNNs are designed to accommodate and take advantage of the sequential nature of such input objects, much like convolutional neural networks accommodate the spatial structure of image inputs.
- The output Y can also be a sequence (such as in language translation), but often is a scalar, like the binary sentiment label of a movie review document.

Structure of a Basic RNN: Input and Output

- RNN takes sequence $X = \{X_1, ..., X_L\}$ as input, with output Y and hidden-layer sequence $\{A_l\}_1^L$
- Activation vector A_l updated using X_l and A_{l-1}, producing prediction O_l for Y

Structure of a Basic RNN: Weight Matrices

- Weight matrices W, U, and vector B for input, hidden-to-hidden, and output layers, respectively

$$A_{lk} = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{lj} + \sum_{s=1}^{K} u_{ks} A_{l-1,s}). \tag{10.16}$$

$$O_l = \beta_0 + \sum_{k=1}^{K} \beta_k A_{lk}. \tag{10.17}$$

Structure of a Basic RNN: Loss Function

- Loss function for regression problems: $(Y - O_L)^2$, using final output O_L
- Parameters learned indirectly via loss function, minimizing sum of squares

$$\sum_{i=1}^{n}(y_i - o_{iL})^2. \tag{10.19}$$

Structure of a Basic RNN: Intermediate Outputs and Applications

- Intermediate outputs O_l needed for some learning tasks with sequence responses
- RNNs used in applications like IMDb sentiment analysis and financial time series forecasting

[Illustration] Schematic of a Simple RNN

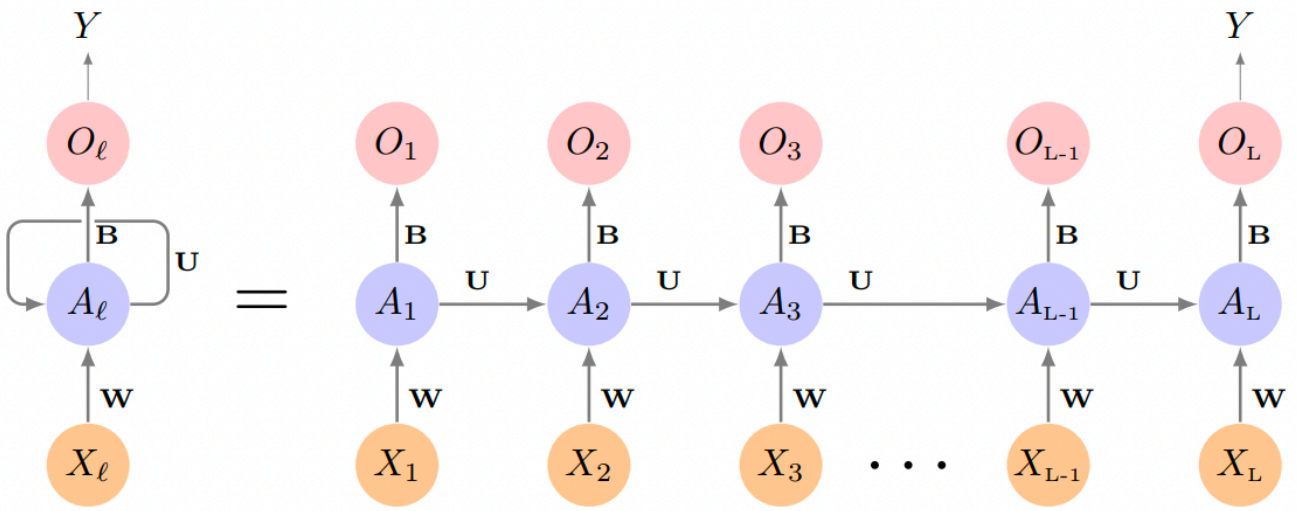


FIGURE 10.12. *Schematic of a simple recurrent neural network. The input is a sequence of vectors* $\{X_\ell\}_1^L$*, and here the target is a single response. The network processes the input sequence* X *sequentially; each* X_ℓ *feeds into the hidden layer, which also has as input the activation vector* $A_{\ell-1}$ *from the previous element in the sequence, and produces the current activation vector* A_ℓ*. The same collections of weights* $\mathbf{W}$*,* $\mathbf{U}$ *and* $\mathbf{B}$ *are used as each element of the sequence is processed. The output layer produces a sequence of predictions* O_ℓ *from the current activation* A_ℓ*, but typically only the last of these,* O_L*, is of relevance. To the left of the equal sign is a concise representation of the network, which is* unrolled *into a more explicit version on the right.*

[Illustration] Depiction of a Sequence of 20 Words

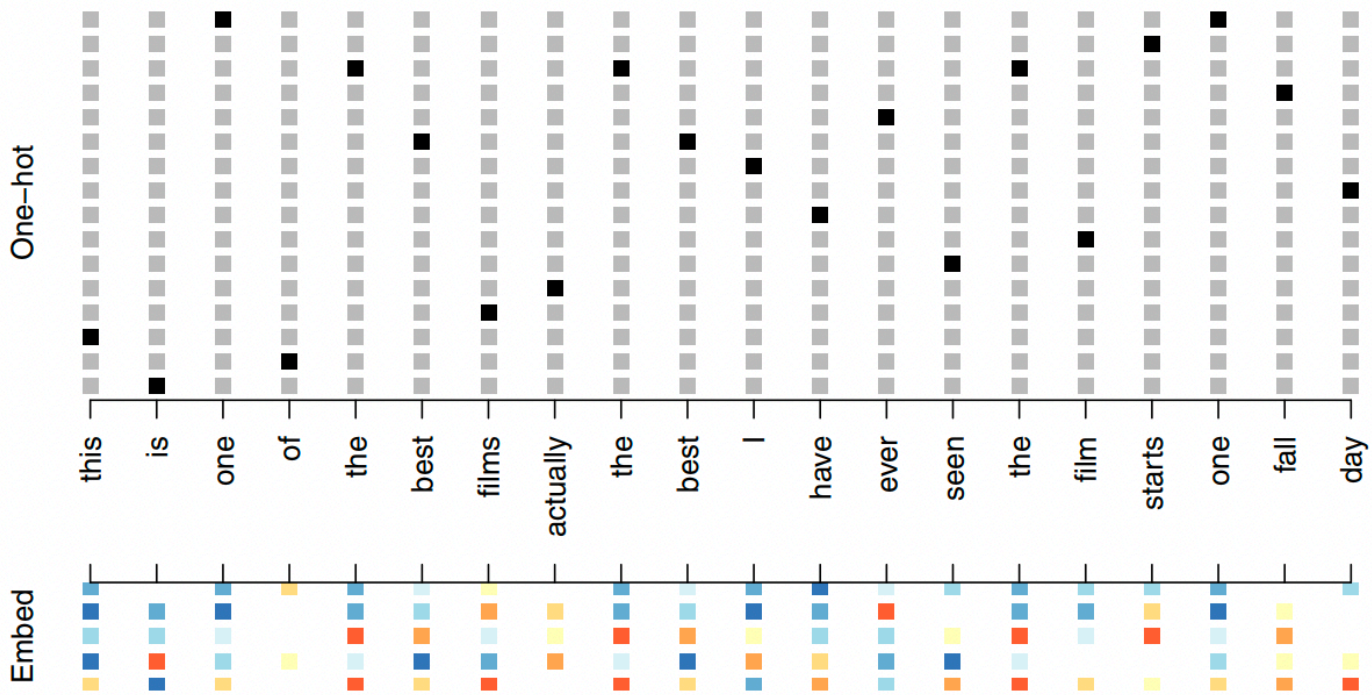


FIGURE 10.13. *Depiction of a sequence of* 20 *words representing a single document: one-hot encoded using a dictionary of* 16 *words (top panel) and embedded in an* m*-dimensional space with* $m = 5$ *(bottom panel).*

Outline

Embedding Space: Word Representation

- Each word in our document is represented by a one-hot-encoded vector with large elements.
- To classify IMDb reviews, we use the sequence of words in the document instead of the bag-of-words model.

Embedding Space: Embedding Matrix

- Each word is represented by a set of m real numbers in a lower-dimensional embedding space (see Figure 10.13).
- The embedding matrix E can be learned from a large corpus of labeled documents or inserted as a precomputed matrix using weight freezing.

Embedding Space: Pretrained Embeddings

- Two widely used pretrained embeddings are word2vec and GloVe.
- If the embedding layer E is learned, it adds an additional $m \times D$ parameters.

LSTM

- More elaborate versions of RNNs use **long term and short term memory (LSTM)** to maintain two tracks of hidden-layer activations, which helps to avoid early signals being washed out by the time they get propagated through the chain to the final activation vector A_L.
- It may improve the performance of fitting the model, but it take a long time to train, which makes exploring many architectures and parameter optimization tedious.
- Despite this added complexity, RNNs provide a rich framework for modeling data sequences, and ongoing advances in architecture, data augmentation, and learning algorithms continue to improve their performance.

Outline

Historical Trading Statistics: Introduction

- Figure 10.14 shows historical trading statistics from the New York Stock Exchange.
- Shown are three daily time series covering the period December 3, 1962 to December 31, 1986[4]

[4]These data were assembled by LeBaron and Weigend (1998) IEEE Transactions on Neural Networks, 9(1): 213–220.

Historical Trading Statistics: Time Series

- Log trading volume: This is the fraction of all outstanding shares that are traded on that day, relative to a 100-day moving average of past turnover, on the log scale.
- Dow Jones return: This is the difference between the log of the Dow Jones Industrial Index on consecutive trading days.
- Log volatility: This is based on the absolute values of daily price movements.

Historical Trading Statistics: Predicting Trading Volume

- Predicting stock prices is a notoriously hard problem, but predicting trading volume based on recent past history is more manageable (and is useful for planning trading strategies).

[Illustration] Trading Statistics (NYSE)

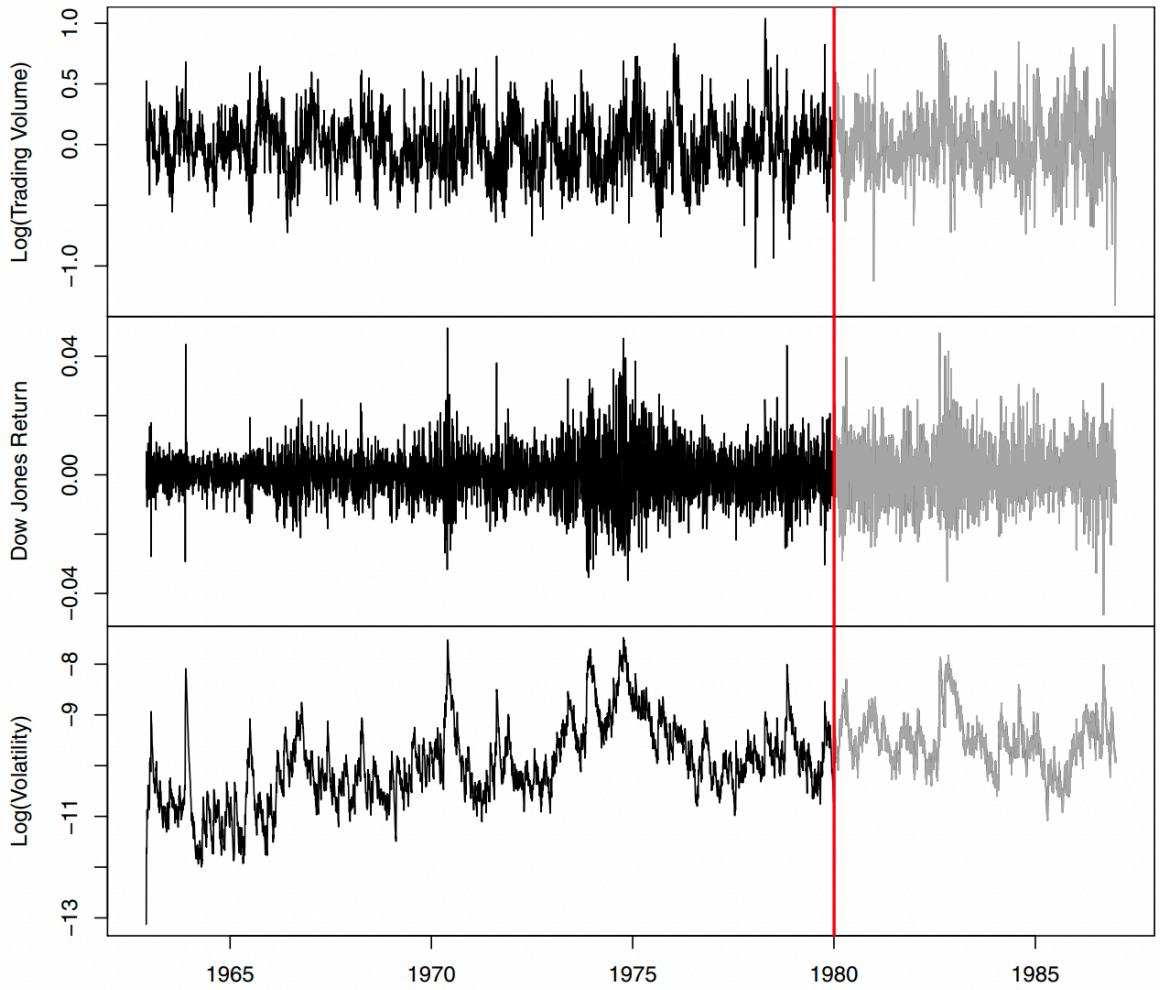


FIGURE 10.14. *Historical trading statistics from the New York Stock Exchange. Daily values of the normalized log trading volume, DJIA return, and log volatility are shown for a 24-year period from 1962–1986. We wish to predict trading volume on any day, given the history on all earlier days. To the left of the red bar (January 2, 1980) is training data, and to the right test data.*

Auto-Correlation: Observations and Time Series

- An observation here consists of the measurements (v_t, r_t, z_t) on day t, in this case the values for log_volume, DJ_return and log_volatility.
- There are a total of $T = 6{,}051$ such triples, each of which is plotted as a time series in Figure 10.14.
- One feature that strikes us immediately is that the day-to-day observations are not independent of each other.

Auto-Correlation: Characteristics

- The series exhibit **auto-correlation** — in this case, values nearby in time tend to be similar to each other.
- This distinguishes time series from other data sets we have encountered, in which observations can be assumed to be independent of each other.

Auto-Correlation: Consideration of Pairs

- To be clear, consider pairs of observations (v_t, v_{t-l}), a lag of l lag days apart.
- If we take all such pairs in the v_t series and compute their correlation coefficient, this gives the autocorrelation at lag l.
- Figure 10.15 shows the autocorrelation function for all lags up to 37, and we see considerable correlation.
- The response variable v_t — log_volume — is also a predictor.

[Illustration] Autocorrelation Function

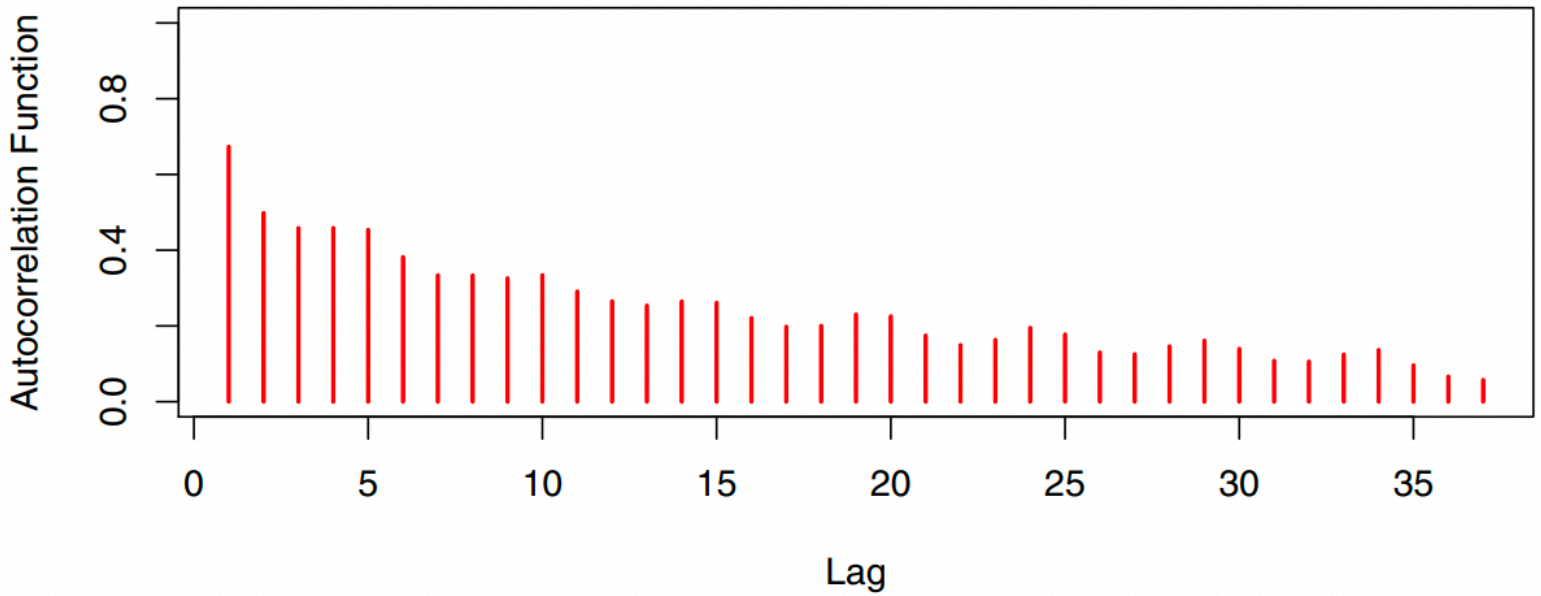


FIGURE 10.15. *The autocorrelation function for* `log_volume`. *We see that nearby values are fairly strongly correlated, with correlations above* 0.2 *as far as 20 days apart.*

RNN Forecaster

- We wish to predict a value v_t from past values $v_{t-1}, v_{t-2}, ...$, and also to make use of past values of the other series$r_{t-1}, r_{t-2}, ...$ and $z_{t-1}, z_{t-2}, ...$
- Although our combined data is quite a long series with 6,051 trading days, the structure of the problem is different from the previous document-classification example.
  - We only have one series of data, not 25,000.
  - We have an entire series of targets v_t, and the inputs include past values of this series.

Lag

- To represent the problem of predicting the value of log volume, we extract short input sequences $X = \{X_1, X_2, ..., X_L\}$ with a predefined length L (called the **lag** in this context), and a corresponding target Y ,where Y is the value of log volume v_t at a single time point t.
- They have the form

$$X_1 = \begin{pmatrix} v_{t-L} \\ r_{t-L} \\ z_{t-L} \end{pmatrix}, X_2 = \begin{pmatrix} v_{t-L+1} \\ r_{t-L+1} \\ z_{t-L+1} \end{pmatrix}, ..., X_L = \begin{pmatrix} v_{t-1} \\ r_{t-1} \\ z_{t-1} \end{pmatrix}, \text{and} Y = v_t. \tag{10.20}$$

- Each input sequence consists of a series of 3-vectors $\{X_l\}_1^L$ each consisting of the three measurements log_volume, DJ_return and log_volatility from day $t - L$, $t - L + 1$, up to $t - 1$.

[Example] NYSE Data

- For the NYSE data, we use the past five trading days to predict the next day's trading volume, hence $L = 5$.
  - We create 6,046 (X, Y) pairs using $T = 6,051$.
  - We fit a model with $K = 12$ hidden units using the 4,281 training sequences derived from the data before January 2, 1980, and achieve an $R^2 = 0.42$ on the test data (Figure 10.14).
  - Using yesterday's value for log_volume as the prediction for today has $R^2 = 0.18$.
  - Figure10.16 shows the observed values of daily log_volume for the test period 1980-1986 in black and the predicted series in orange, which appears to have a good correspondence.
  - In forecasting the value of log_volume in the test period, we use the test data itself in forming the input sequences X, but this is not considered cheating as we are always using past data to predict the future.

[Illustration] RNN Forecast of log_volume

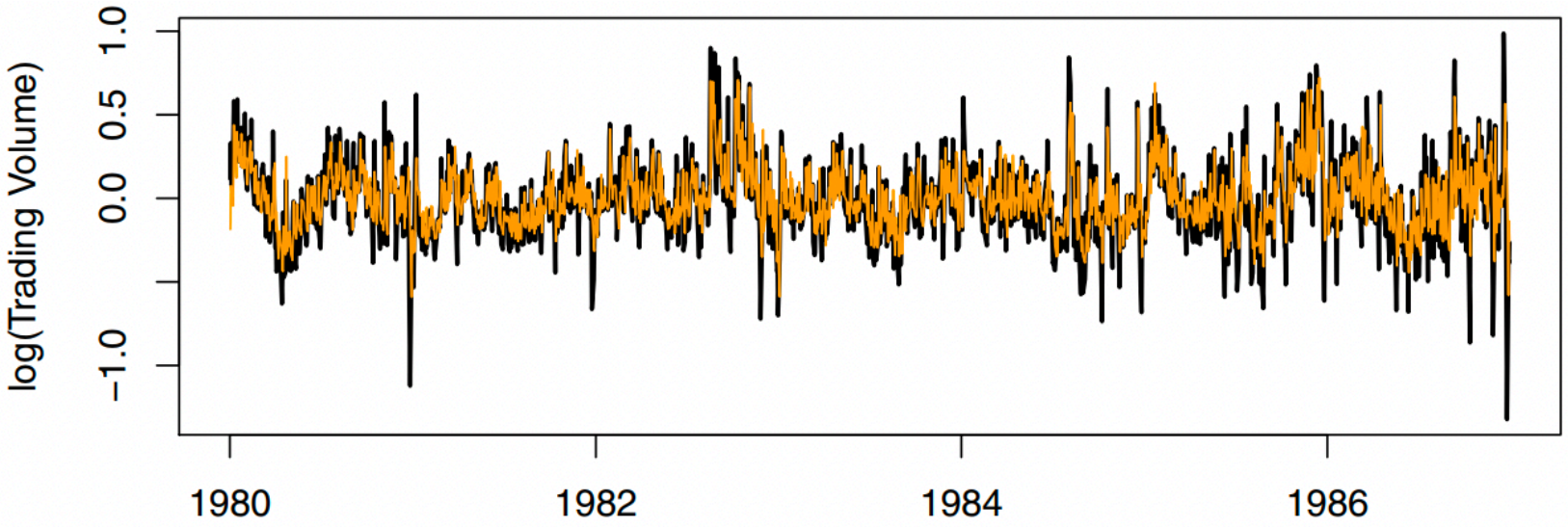


FIGURE 10.16. *RNN forecast of* `log_volume` *on the* `NYSE` *test data. The black lines are the true volumes, and the superimposed orange the forecasts. The forecasted series accounts for 42% of the variance of* `log_volume`.

Autoregression

- RNN and traditional autoregression (AR) linear model share similarities
- AR model constructs response vector y and predictor matrix M for least squares regression

$$y = \begin{bmatrix} v_{L+1} \\ v_{L+2} \\ v_{L+3} \\ \vdots \\ v_T \end{bmatrix}, M = \begin{bmatrix} 1 & v_L & v_{L-1} & \cdots & v_1 \\ 1 & v_{L+1} & v_L & \cdots & v_2 \\ 1 & v_{L+2} & v_{L+1} & \cdots & v_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_{T-1} & v_{T-2} & \cdots & v_{T-L} \end{bmatrix}. \quad (10.21)$$

- Predictors for v_t are previous L values of the same series
- Fitting AR(L) model includes lagged versions of other variables
- For NYSE data, include lagged versions of DJ_return and log_volatility, resulting in $3L + 1$ columns in predictor matrix M

AR Model vs. RNN

- AR model with $L = 5$: test $R^2 = 0.41$, slightly inferior to RNN's 0.42
- RNN and AR models use the same response Y and input sequences X
- RNN processes sequence with same weights W, while AR model flattens inputs
- RNN includes hidden layer activations A_l, introducing nonlinearity
- RNN has 205 parameters, compared to 16 for AR(5) model

Model Improvements

- Extend AR model by using lagged predictors in a neural network
  - Achieved test $R^2 = 0.42$, slightly better than linear AR
- Include day of week variable for better performance
  - AR model improved to $R^2 = 0.46$, nonlinear AR model to $R^2 = 0.47$
- LSTM extension of RNN yields small improvements, up to 1% in R^2

Outline

Summary of RNNs

- RNNs have many variations and enhancements, such as using a one-dimensional convolutional neural network or having additional hidden layers.
- Bidirectional RNNs can scan sequences in both directions, and in Seq2Seq learning, RNNs can be used for language modeling and translation.
- However, algorithms used to fit RNNs can be complex and computationally costly.
- Fortunately, good software makes specifying and fitting these models relatively painless, and state-of-the-art architectures developed by skilled engineers and trained on massive computational and data resources are used in many daily-life applications such as Google Translate.

Outline

When to Use Deep Learning

- Deep learning has shown impressive performance in various tasks, such as image classification, machine diagnosis of medical images, speech and language translation, forecasting, and document modeling.
- However, the question of whether to discard older tools and use deep learning on every problem with data remains.
  - To address this question, we revisit the Hitters dataset from Chapter 6.
  - In the Hitters dataset, a regression problem aims to predict the Salary of a baseball player in 1987 using his performance statistics from 1986.
  - Three methods are used for fitting a regression model to the data (next page).

Comparison of Models

- Table 10.2 compares the results of three models, showing similar performance.
- Mean absolute error and test R^2 are reported for each method, all respectable.
- Linear models were easier to obtain and understand than the neural network, which is essentially a **black box**.
- The **lasso** selected 12 of the 19 variables in making its prediction, following the **Occam's razor** principle to pick the simplest method when faced with similar performance.

Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network	1409	257.4	0.54

TABLE 10.2. *Prediction results on the* `Hitters` *test data for linear models fit by ordinary least squares and lasso, compared to a neural network fit by stochastic gradient descent with dropout regularization.*

Simpler Model

- Exploration with Lasso:
  - After exploration with the lasso model, an even simpler model with four variables was identified.
  - The linear model was refit with these four variables to the training data (relaxed lasso) and achieved the lowest test mean absolute error of 224.8.
- Avoiding Selection Bias:
  - To avoid selection bias, the model was refit on the test data instead of presenting the summary table with coefficients and p-values from the selected model on the training data.

	Coefficient	Std. error	t-statistic	p-value
Intercept	-226.67	86.26	-2.63	0.0103
Hits	3.06	1.02	3.00	0.0036
Walks	0.181	2.04	0.09	0.9294
CRuns	0.859	0.12	7.09	< 0.0001
PutOuts	0.465	0.13	3.60	0.0005

TABLE 10.3. *Least squares coefficient estimates associated with the regression of* Salary *on four variables chosen by lasso on the* Hitters *data set. This model achieved the best performance on the test data, with a mean absolute error*

Consider Simpler Models

- Powerful Tools:
  - Powerful tools available: neural networks, random forests, boosting, SVMs, etc.
- Don't Forget Simpler Models:
  - Don't forget linear models and simpler variants.
- Advantages of Simpler Models:
  - Simpler models can be easier to fit, understand, and less fragile.
- Performance/Complexity Tradeoff:
  - Try simpler models and weigh performance/complexity tradeoff.
- When to Use Deep Learning:
  - Use deep learning for large training sets and when interpretability is not crucial.

Fitting a Neural Network

- Nonconvex problem with multiple solutions
- Nonlinear least squares problem: $\min_{\{w_k\}_1^K, \beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - f(x_i))^2$
- Nested parameters and hidden units' symmetry
- Strategies: **slow learning** (gradient descent) and **regularization** (lasso/ridge penalties)
- Objective: $R(\theta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$
- Gradient descent: Iteratively find small parameter changes to reduce objective
- Goal: Reach minimum of objective, potentially a good local minimum

[Illustration] Gradient Descent for One-Dimensional ϑ

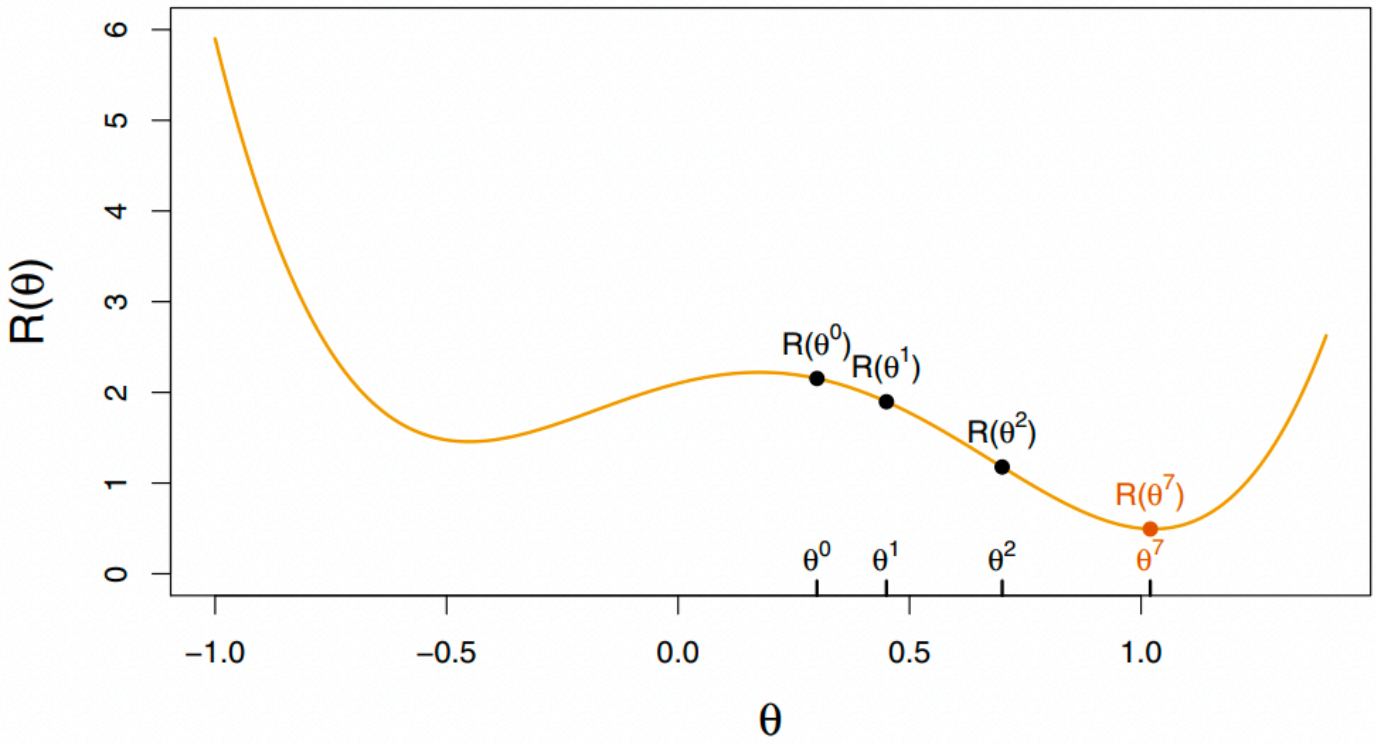


FIGURE 10.17. *Illustration of gradient descent for one-dimensional θ. The objective function $R(\theta)$ is not convex, and has two minima, one at $\theta = -0.46$ (local), the other at $\theta = 1.02$ (global). Starting at some value θ^0 (typically randomly chosen), each step in θ moves downhill — against the gradient — until it cannot go down any further. Here gradient descent reached the global minimum in 7 steps.*

Backpropagation and Gradient Descent

- Objective: Minimize $R(\theta)$ using gradient descent
- Gradient: $\nabla R(\theta^m) = \frac{\partial R(\theta)}{\partial \theta} \mid_{\theta=\theta^m}$
- Update rule: $\theta^{m+1} \leftarrow \theta^m - \rho \nabla R(\theta^m)$
- Chain rule simplifies gradient calculation for complex networks
- One term of the sum:
  $R_i(\theta) = \frac{1}{2}(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}))^2$
- Partial derivatives: $\frac{\partial R_i(\theta)}{\partial \beta_k} = -(y_i - f_\theta(x_i)) \cdot g(z_{ik})$ and
  $\frac{\partial R_i(\theta)}{\partial w_{kj}} = -(y_i - f_\theta(x_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}$, with $z_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}$
- Backpropagation: Assigns fraction of residual to each parameter via activation functions

Regularization and Stochastic Gradient Descent

- Accelerating gradient descent with Stochastic Gradient Descent (SGD)
- SGD: Sample minibatch of observations for each gradient step
- Regularization: Essential to avoid overfitting, e.g., ridge regularization on weights
- Augmented objective function:
  $R(\theta; \lambda) = -\sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} \log(f_m(x_i)) + \lambda \sum_j \theta_j^2$
- Parameter λ: Set to a small value or found using validation-set approach
- SGD enforces approximately quadratic regularization
- Early stopping as an additional form of regularization

[Illustration] Evolution of Training & Validation Errors

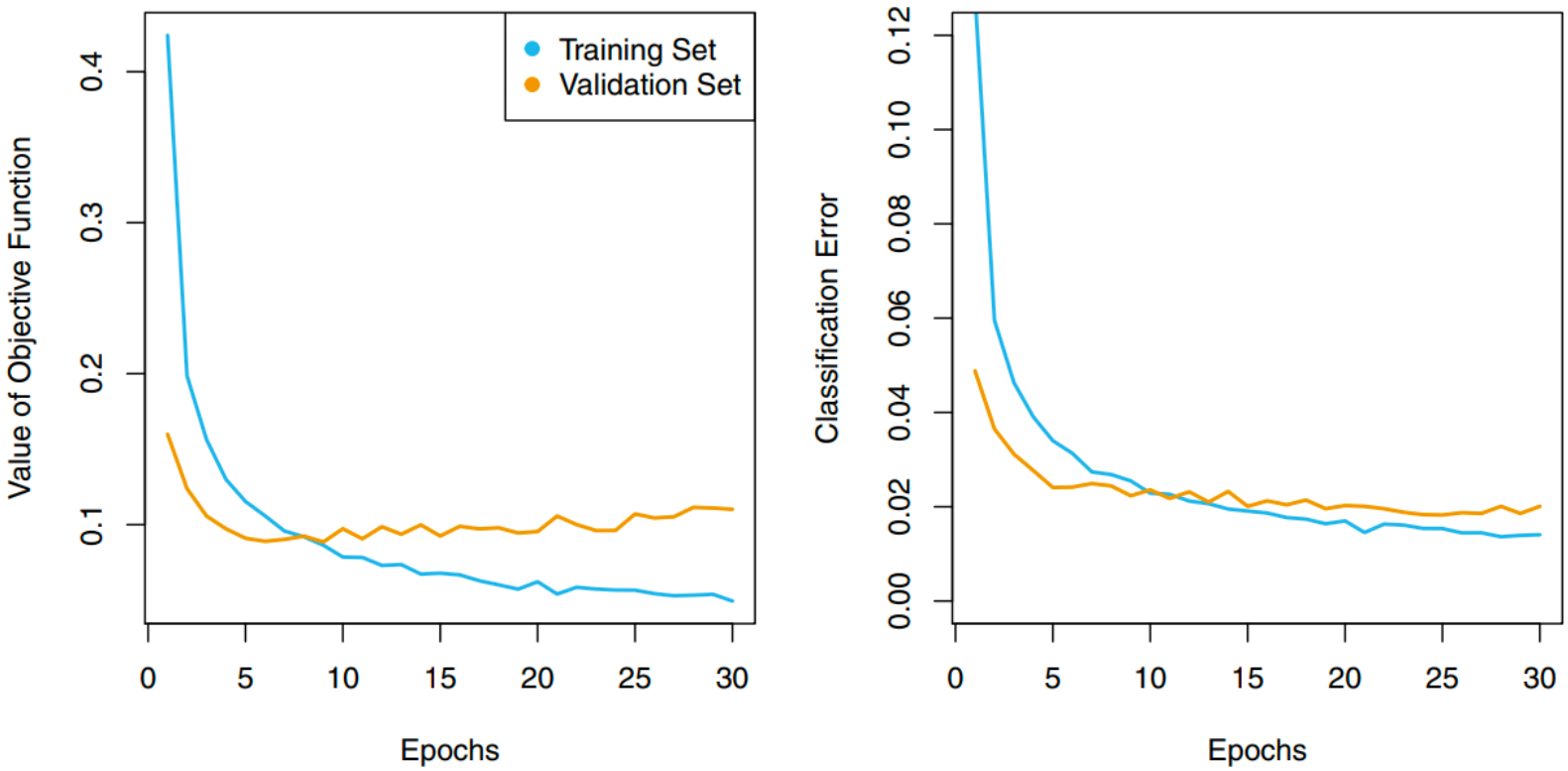


FIGURE 10.18. *Evolution of training and validation errors for the* `MNIST` *neural network depicted in Figure 10.4, as a function of training epochs. The objective refers to the log-likelihood (10.14).*

Dropout Learning

- The second row in Table 10.1 is labeled **dropout**.
  - This is a relatively new and efficient form of regularization, similar in some respects to ridge regularization.
  - Inspired by random forests (Section 8.2), the idea is to randomly remove a fraction ϕ of the units in a layer when fitting the model. Figure 10.19 illustrates this.
  - This is done separately each time a training observation is processed. The surviving units stand in for those missing, and their weights are scaled up by a factor of $1/(1 - \phi)$ to compensate.
  - This prevents nodes from becoming over-specialized, and can be seen as a form of regularization.
  - In practice dropout is achieved by randomly setting the activations for the "dropped out" units to zero, while keeping the architecture intact.

[Illustration] Dropout Learning

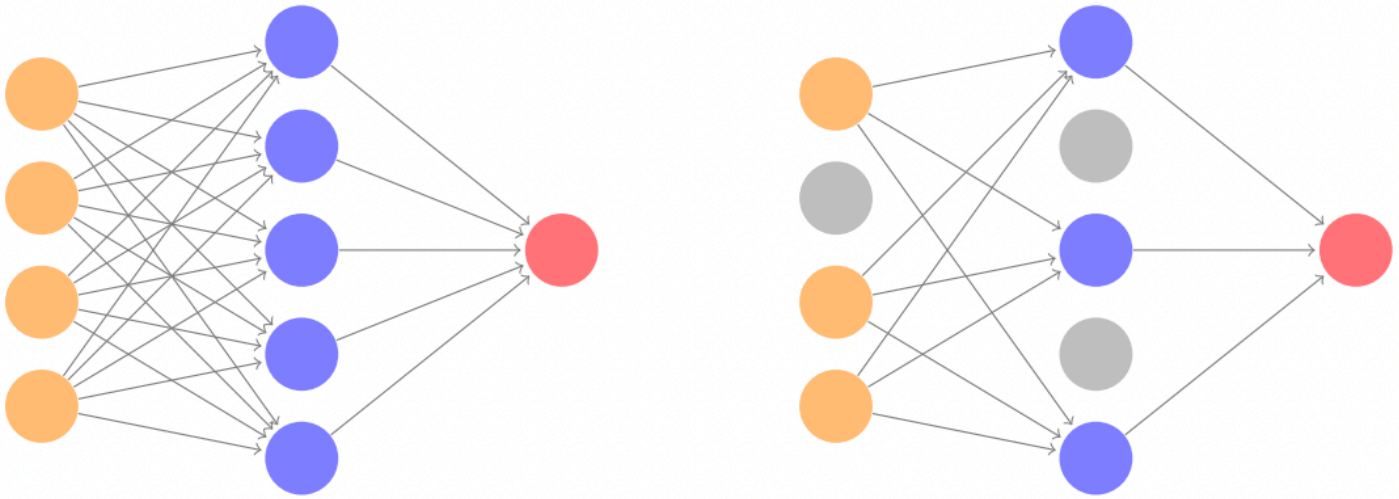


FIGURE 10.19. *Dropout Learning. Left: a fully connected network. Right: network with dropout in the input and hidden layer. The nodes in grey are selected at random, and ignored in an instance of training.*

Network Tuning: Introduction

- The network in Figure 10.4 is considered to be relatively straightforward; it nevertheless requires a number of choices that all have an effect on the performance.

Network Tuning: Hidden Layers and Units

- **The number of hidden layers, and the number of units per layer**
- Modern thinking is that the number of units per hidden layer can be large, and overfitting can be controlled via the various forms of regularization.

Network Tuning: Regularization Parameters

- **Regularization tuning parameters**
- These include the dropout rate ϕ and the strength λ of lasso and ridge regularization, and are typically set separately at each layer.

Network Tuning: Stochastic Gradient Descent

- **Details of stochastic gradient descent**
- These includes the batch size, the number of epochs, and if used, details of data augmentation (Section 10.3.4).

Network Tuning: Importance of Choices

- Choices such as these can make a difference.
- In preparing this MNIST example, we achieved a respectable 1.8% misclassification error after some trial and error.
- Finer tuning and training of a similar network can get under 1% error on these data, but the tinkering process can be tedious, and can result in overfitting if done carelessly.

Interpolation and Double Descent

- Bias-variance trade-off: Interpolating training data (zero training error) can lead to high test error
- Double descent: In some cases, interpolating models perform better than slightly less complex ones
- Test error: U-shaped before interpolation threshold, then descends as model flexibility increases
- Example: Simulated $n = 20$ observations from $Y = \sin(X) + \epsilon$, $X \sim U[-5, 5]$, $\epsilon \sim N(0, \sigma^2)$, $\sigma = 0.3$
- Fitted natural spline with d degrees of freedom; selected d with smallest $\sum_{j=1}^{d} \hat{\beta}_j^2$ (minimum-norm solution; Figure 10.21)

[Illustration] Double Descent Phenomenon

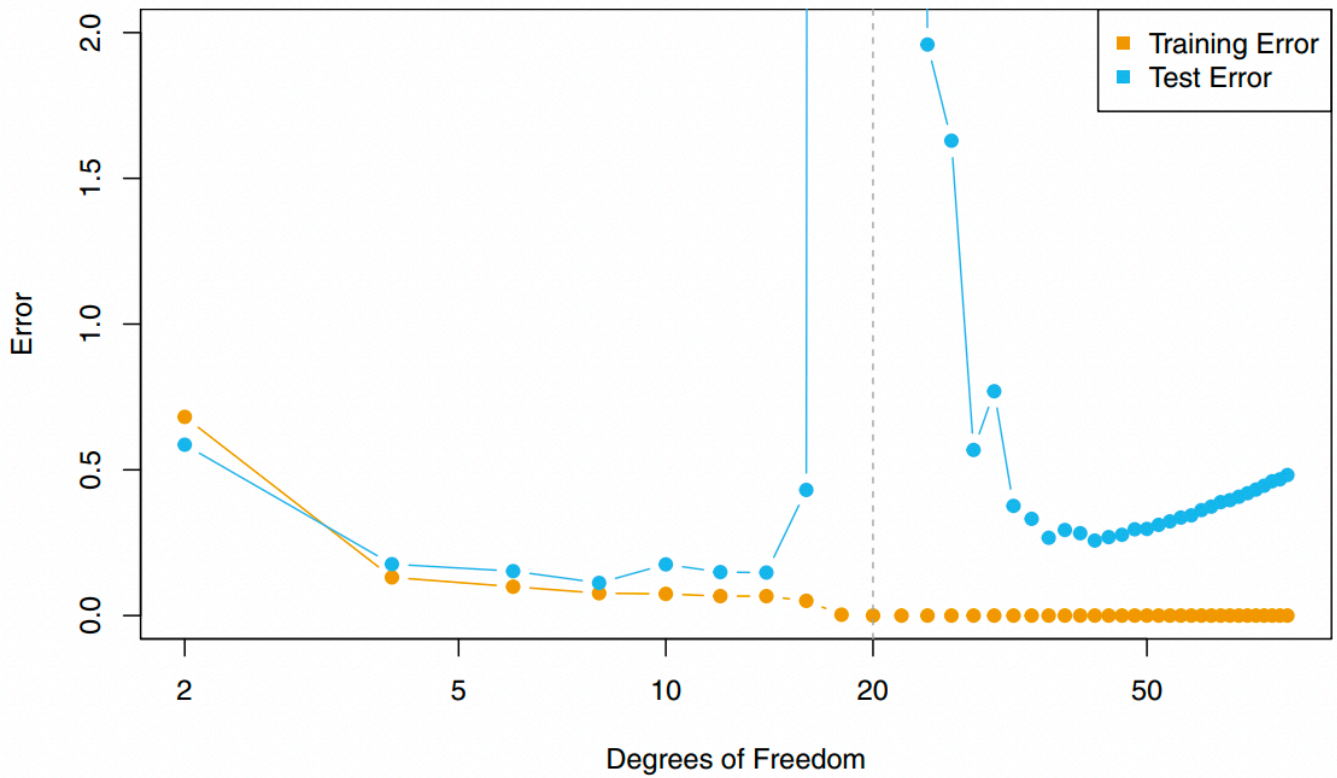


FIGURE 10.20. *Double descent phenomenon, illustrated using error plots for a one-dimensional natural spline example. The horizontal axis refers to the number of spline basis functions on the log scale. The training error hits zero when the degrees of freedom coincides with the sample size $n = 20$, the "interpolation threshold", and remains zero thereafter. The test error increases dramatically at this threshold, but then descends again to a reasonable value before finally increasing again.*

[Illustration] $\hat{f}_d(X)$,$f(X)$, and Training Data Points

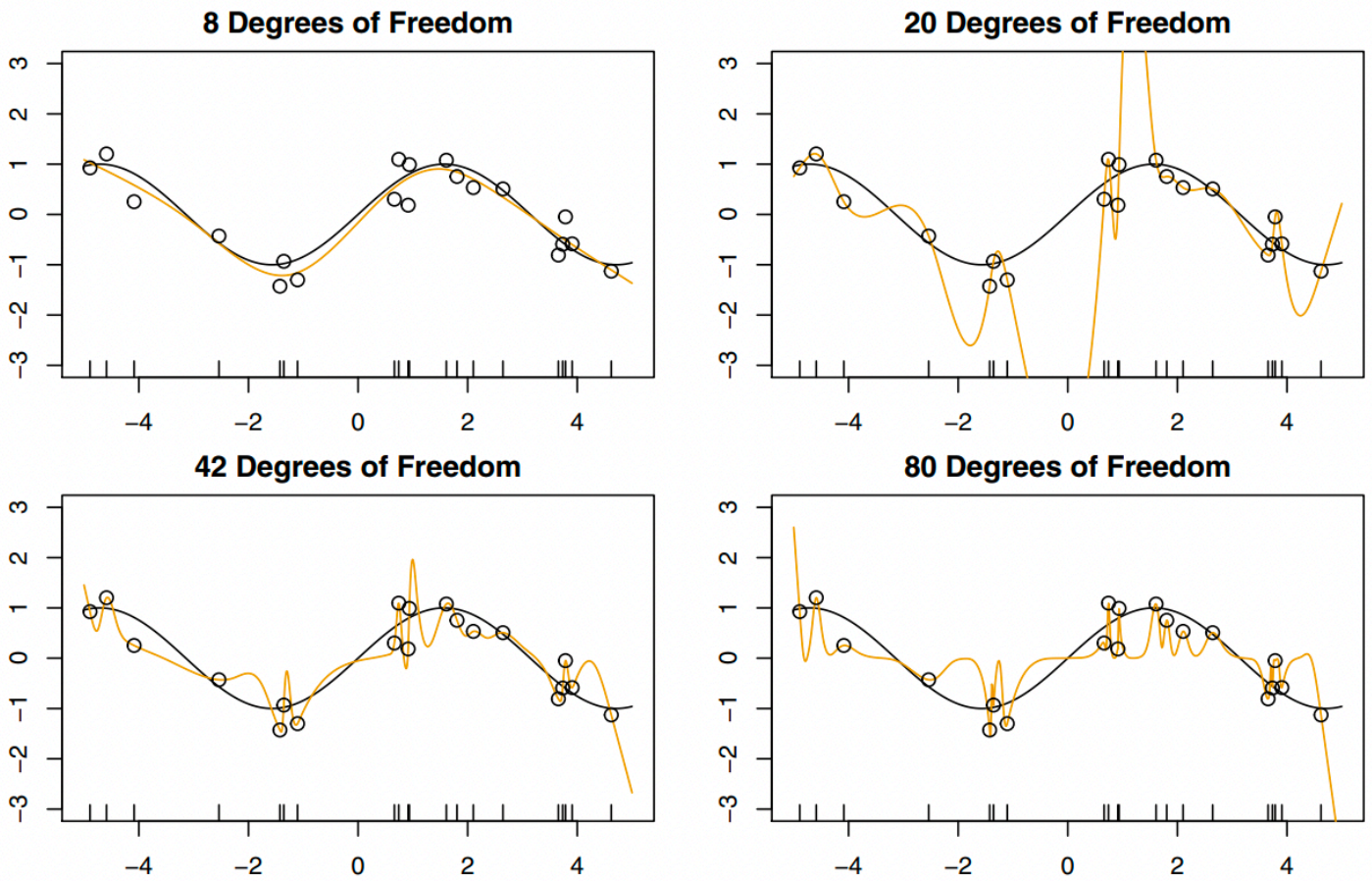


FIGURE 10.21. *Fitted functions $\hat{f}_d(X)$ (orange), true function $f(X)$ (black) and the observed 20 training data points. A different value of d (degrees of freedom) is used in each panel. For $d \geq 20$ the orange curves all interpolate the training points, and hence the training error is zero.*

Huge Number of Parameters Can Be Beneficial

- Minimum-norm natural splines: smoother with more degrees of freedom (e.g., $d = 42$ and $d = 80$)
- Neural networks with many parameters can achieve good results and zero training error
- Especially effective in high signal-to-noise ratio problems (e.g., image recognition, language translation)
- Stochastic gradient descent helps select "smooth" interpolating models with good test-set performance

Important Points and Double-Descent Phenomenon

- Double-descent does not contradict bias-variance trade-off
- Minimum-norm natural spline with $d = 42$ has lower variance than with $d = 20$
- Most statistical learning methods do not exhibit double descent
- Regularization methods do not interpolate data, avoiding double descent
- Maximal margin classifiers and SVMs achieve good test error with zero training error
- Overparametrized neural networks can fit to zero training error, but it's not always optimal
- Signal-to-noise ratio is important when deciding to fit to zero error
- Ridge regularization and early stopping help prevent overfitting and improve test-set performance

Summary

- Although double descent can sometimes occur in neural networks, we typically do not want to rely on this behavior.
- Moreover, it is important to remember that the bias-variance trade-off always holds (though it is possible that test error as a function of flexibility may not exhibit a U-shape, depending on how we have parametrized the notion of "flexibility" on the x-axis).

For Further Reading I

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: