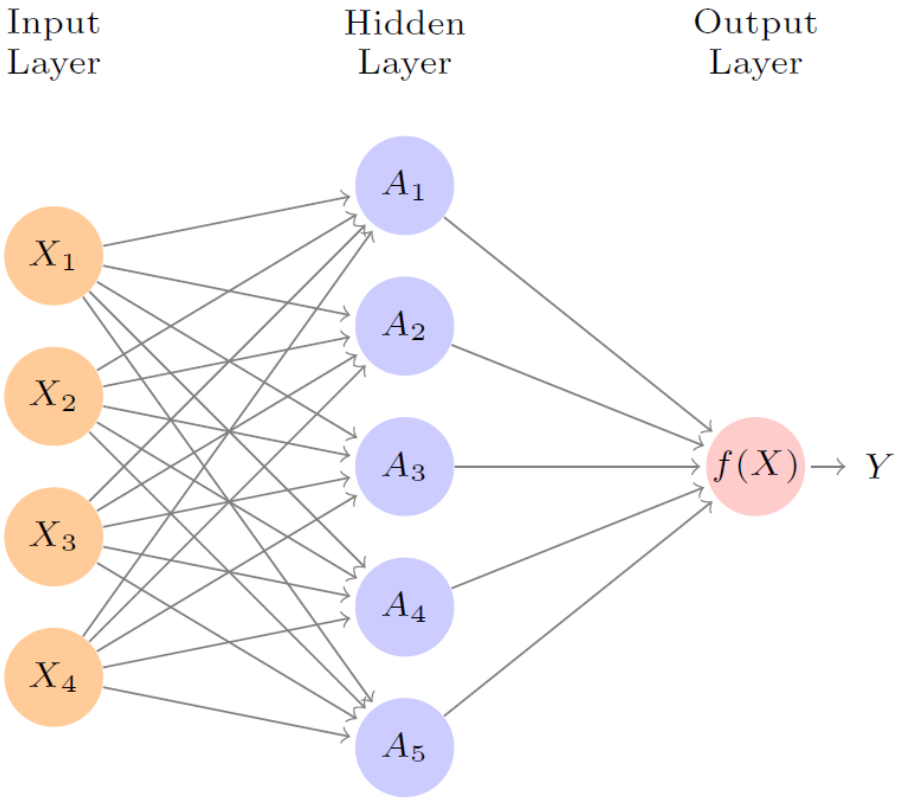
Chapter 10: Deep Learning

- This chapter will focus on neural networks as the primary example of deep learning.
- First developed in the 1980's it has seen a resurgence since 2010.
- A learning method developed in different fields, statistics and artificial intelligence.
- The method generates a derived variable from linear functions of the predictors.
- These linear functions are then used as inputs to non-linear functions to predict outcomes.

Single Layer Neural Networks

- This figure shows a simple feed-forward neural network
- It has four predictors (p=4), X_1,..,X_4 make up the input layer.
- Each predictor feeds into each of the K (5) hidden units, which produce activations, A_1,..,A_5.
- The activations then feed into the output layer.



Single Layer Neural Networks

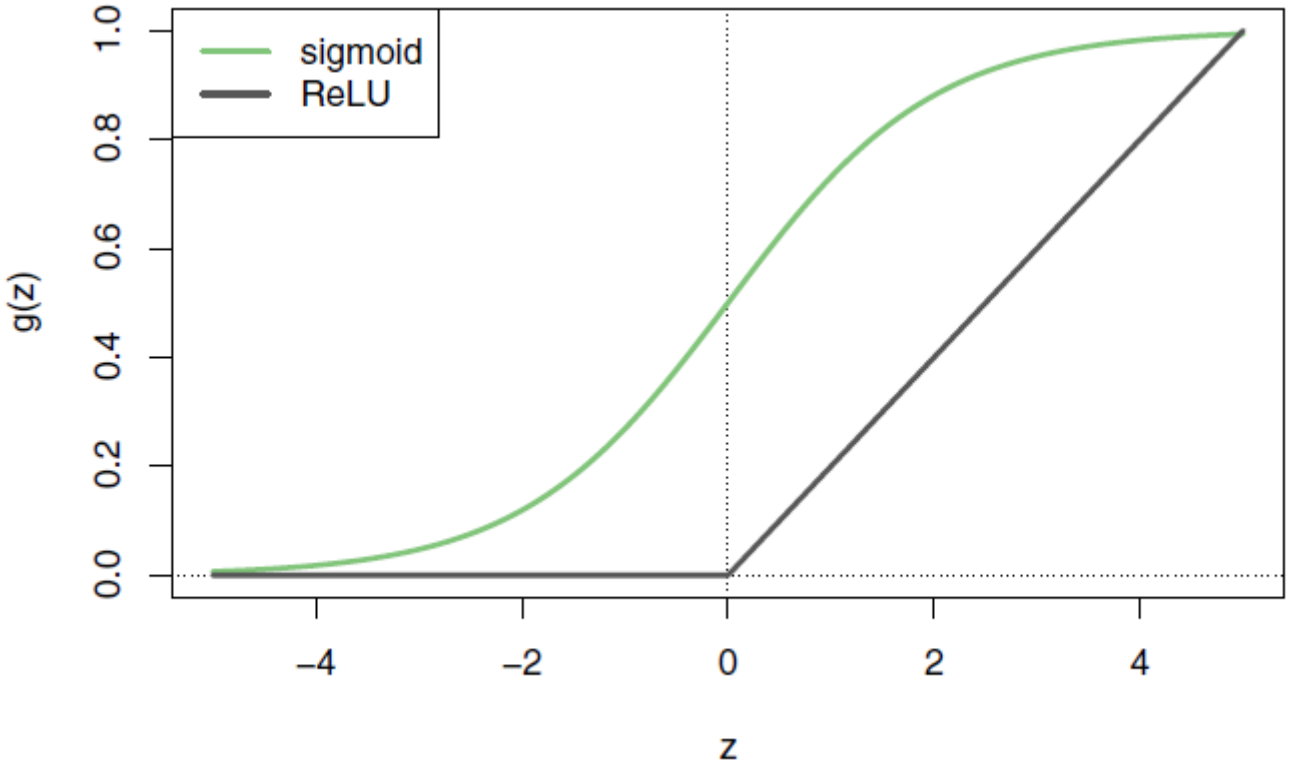
- The general model for the response is, $f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$
- The activation functions, $h_k(X)$, are modeled by a pre-defined nonlinear function, $g\left(w_{k0} + \sum_{k=1}^{p} w_{kj} X_j\right) = h_k(X)$. For the previous example that means there are a total of 30 parameters to estimate.
- In the past a sigmoid activation function was used, $g(z) = \frac{1}{1+e^{-z}}$. Currently the ReLU (rectified linear unit) is favored,

$$g(z) = (z)_+ = \begin{cases} 0 \; if \; z < 0 \\ z \; otherwise \end{cases}$$

Activation Functions

If the activation function was linear then the entire model would be linear.

The non-linear activation functions allow the model to capture complex nonlinearities and interaction effects.



Multilayer Neural Networks

This example is for digital recognition of handwritten numbers. Input is a 28 × 28 grid of 784 pixels.

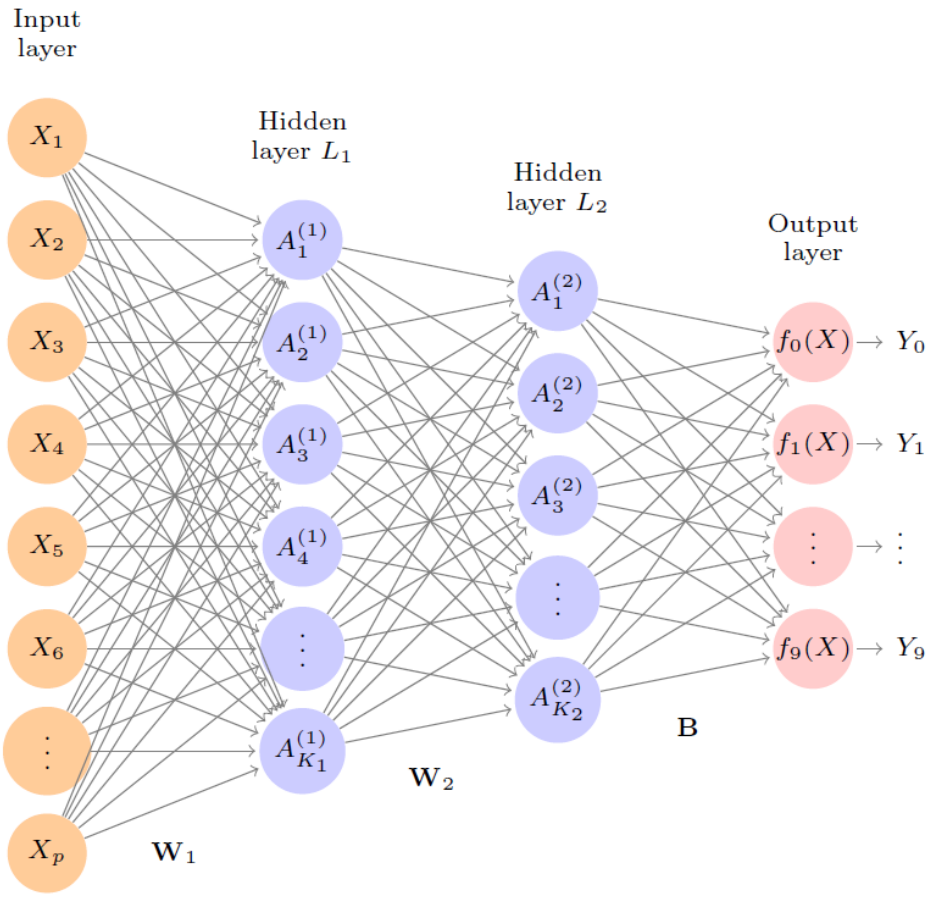
There are two hidden layers, L_1 (256 units) and L_2 (128 units), and 10 output variables, 0, 1…, 9.

The first activation layer is like the single layer neural network, $A_k^{(1)} = h_k^{(1)}(X)$ for k= 1 … K_1.

The second layer treats the activations, $A_k^{(1)}$, as inputs, $A_k^{(2)} = h_l^{(2)}(X) = g\left(w_{l0}^{(2)} + \sum_{k=1}^{K_1} w_{lk}^{(2)} A_k^{(1)}\right)$ for l= 1 … K_2.

W_1 is the matrix of weights (coefficients) that feed input to the first layer. The matrix has 785 (784 pixels +intercept) × 256 (grey scales) = 200,960 elements.

The output layer takes the second layer activations and generates the 10 outputs, $Z_m = \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} A_l^{(2)}$



Multilayer Neural Networks: outputs

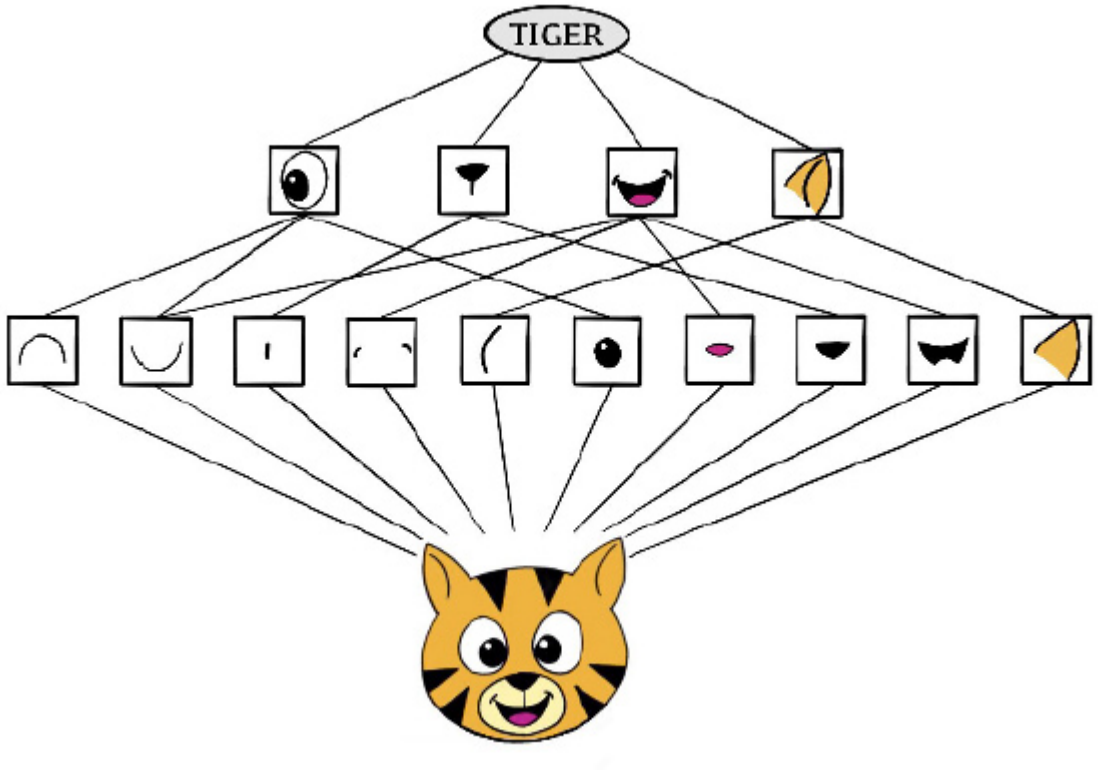
- If the outputs were quantitative variables then $f_m(X) = Z_m$.
- However, if we are classifying inputs to categories we change the Z_m to probabilities, $f_m(X) = \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}}$, called the softmax activation function.
- For a quantitative response the model is trained by minimizing the MSE, $\sum_{i=1}^{n}(y_i - f(x_i))^2$.
- For a qualitative variable we will minimize the negative of a multinomial log-likelihood, $-\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log(f_m(x_i))$. Here $y_{im} = 1$, if $y_{im} = m$ and 0 otherwise.
- The neural network has a large number of parameters and thus needs regularization by either of two methods: ridge regularization or dropout regularization.

Convolutional Neural Networks (CNN)

- The technique will be illustrated with a categorical response (animal name) and digital image inputs. There are a total of 100 classes of animals, e.g. beaver (aquatic animal).
- Each image in the CIFAR100 database has a resolution of 32×32 pixels. Associated with each pixel are three, eight-bit numbers representing colors, red, blue, and green.
- These numbers are organized in a three dimensional array called a feature map.
- The first two axes of the array are spatial, each with 32 dimensions. The third axis is the channel axis representing the three colors.

Convolutional Neural Networks

- The network takes in small, local features.
- These are then convoluted to combine the small features into larger recognizable parts.
- The animal can then be identified from these recognizable parts.

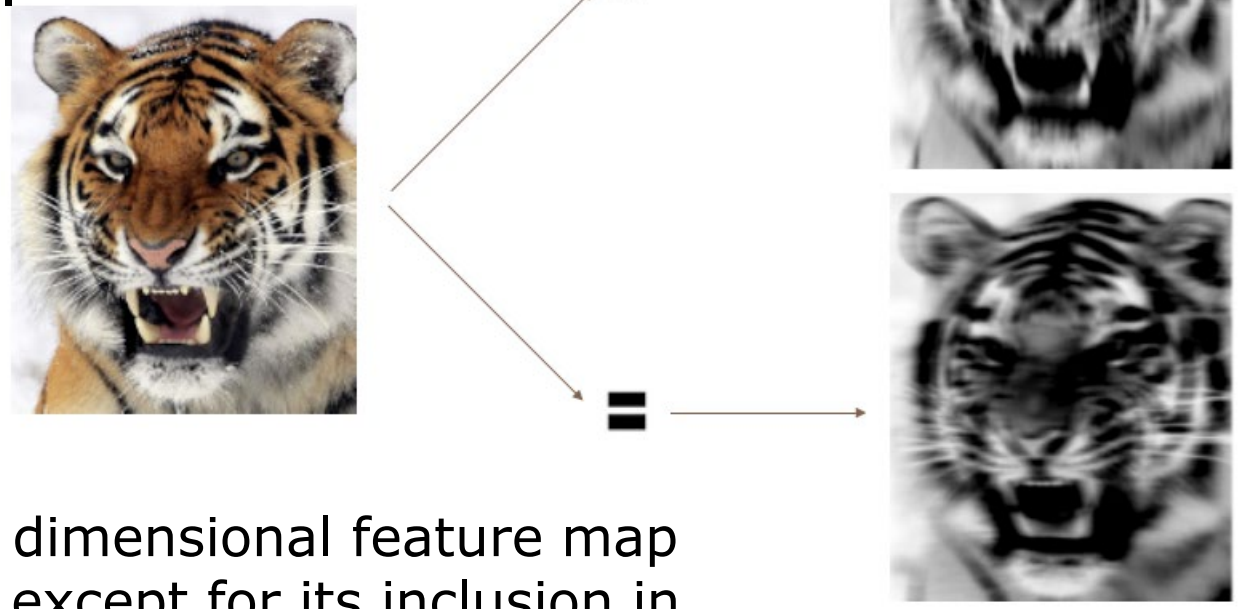


Convolution Layers

- The original image data is a 4×3 matrix, $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$.
- The convolutional filter is, $\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$.
- The convolved image is, $\begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$
- The convolved image has now added pieces of data from a close neighbor which for image analysis means a nearby piece of anatomy.

Convolution Filters

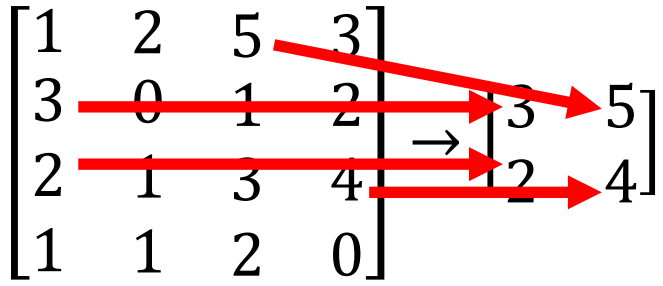
- Application (Tiger) of 15×15 filters with either a narrow strip of horizontal 1's (the rest 0's) or vertical 1's.
- The original picture is the input and the convolved images are the first hidden layer.
- For the CIFAR100 database each color channel has a 32×32 feature map. A single convolution filter will also have 3, 3×3 channels.



- The output of these convolutions is a two dimensional feature map and no further reference to color is made except for its inclusion in the convolution.
- *K* filters will produce *K* two-dimensional feature maps.
- Together they are a three-dimensional map with *K* channels.
- An ReLU activation function may be applied to the convolved image.
- This would be treated as a separate layer – called a detector layer.

Pooling Layers

- These condense a large image into a smaller summary image.
- One method is max pooling, applying a summary to non-overlapping 2×2 blocks. This will reduce the size of the image by a factor of 2 in each dimension.
- Example:



Architecture of a Convolutional Neural Network

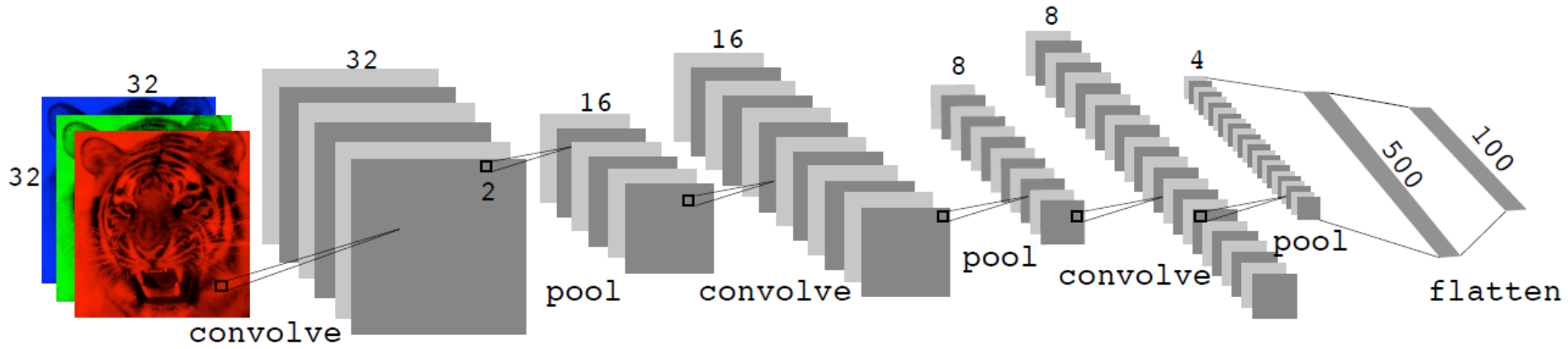


FIGURE 10.8. *Architecture of a deep CNN for the* `CIFAR100` *classification task. Convolution layers are interspersed with* 2×2 *max-pool layers, which reduce the size by a factor of 2 in both dimensions.*

- There are usually multiple rounds of convolution and pooling.
- Since the pooling layer decreases the size of each feature, this is usually followed by application of an increased number of filters that produce a greater number of channels.

Architecture of a Convolutional Neural Network

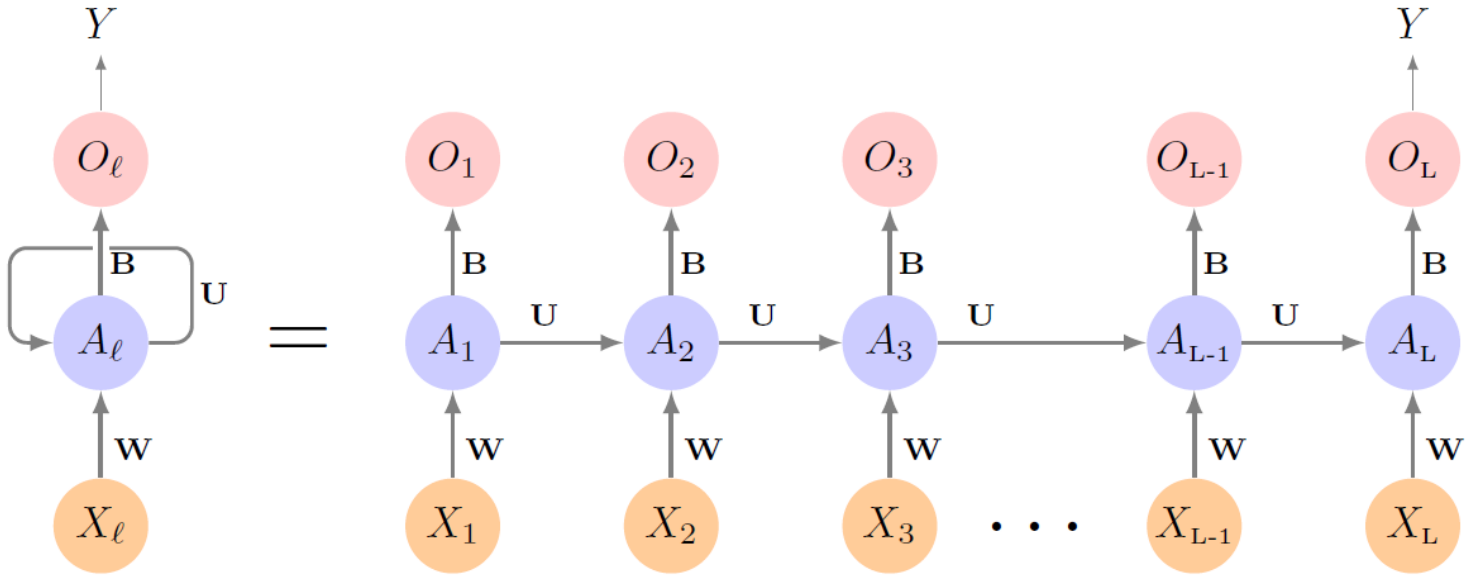
- To keep the convolved maps the same dimension as the original maps “padding” is applied.
- When the channel feature map has been reduced to a few pixels the image is “flattened”.
- They are fed into one or more fully connected layers before reaching the softmax output layer which produces 100 probabilities for the 100 different classes.

Recurrent Neural Networks (RNN)

- Examples: a movie review on the IMDb database. Can we determine if it is a positive of negative review?
- The data must first be “featurized”. That is we have to define a set of features from the text.
- Bag-of-words approach. Take, say, distinct pairs of words. So a due like “Blissfully long” might be indicative of a positive review while “Blissfully short” might be a negative review.
- Treat the document as a sequence assuming that their order matters.
- Examples of sequential data might include documents, time series like temperature, rainfall, and air quality, financial time series.

Recurrent Neural Networks

- A word sequence might be broken into words and stored in a vector, $\boldsymbol{X}=\{X_1, \ldots, X_L\}$, where each X_l is a word.



The sequence above is analyzed one vector at a time. Each unit produces activation output, A_l, to a hidden layer. The units are then updated with their input, X_l, and the activation output from the unit before them, A_{l-1}. Each unit produces output, O_l, although only the last output may be of interest.

More generally each input vector may be composed of p components, $X_l^T = (X_{l1}, .., X_{lp})$. The hidden layer consists of K units (these may be the updates), $A_l^T = (A_{l1}, .., A_{lK})$. Thus, **W** is the $K\times(p+1)$ shared weights, w_{kj}, for input layer. **U** is the $K\times K$ matrix of weight for the hidden-to-hidden layer, u_{ks}. Finally, **B** is the $K+1$ vector of weight for the output layer, β_k.

Recurrent Neural Networks

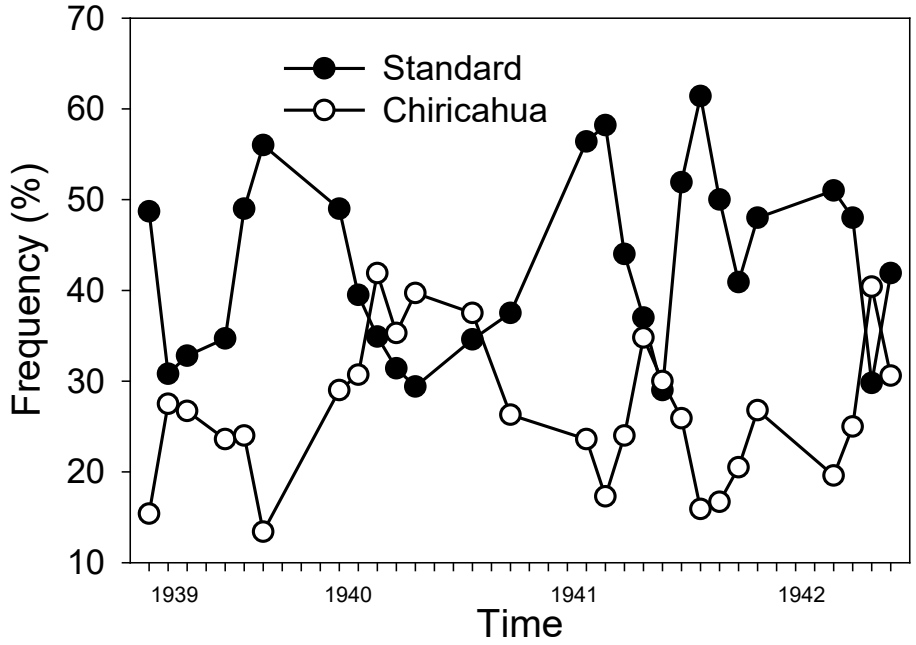
- The activations are calculated as,

$$A_{lk} = g\left(w_{k0} + \sum_{j=1}^{p} w_{jk} X_{lj} + \sum_{s=1}^{K} u_{ks} A_{l-1,s}\right)$$

- The output is, $O_l = \beta_0 + \sum_{k=1}^{K} \beta_k A_{lk}$
- Note that the same weights are used as the information is processed along the sequence, e.g. the weights in **W**, **U** and **B** are not functions of *l*.
- For regression problems the loss function is $(Y - O_L)^2$. The other outputs are not used (unless the output is vector valued).
- However, O_L does depend on all the output due to the sequential processing of the hidden layer outputs.

Time Series Forecasting

- Observations of one or more variables that are taken at regular time intervals.
- Examples include chromosome inversion frequencies in *Drosophila pseudoobscura* or financial data like trading volumes (v_t), the Dow Jones average (r_t) or market volatility (z_t).

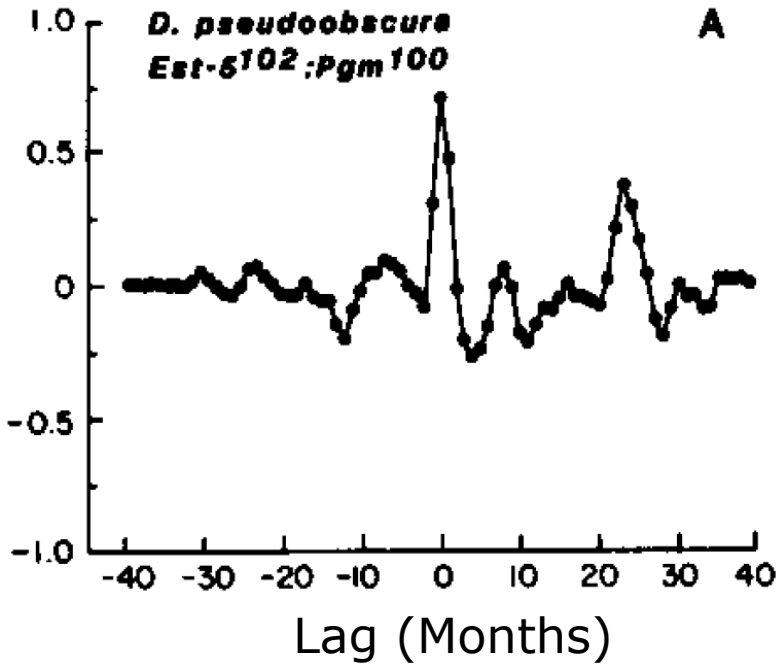


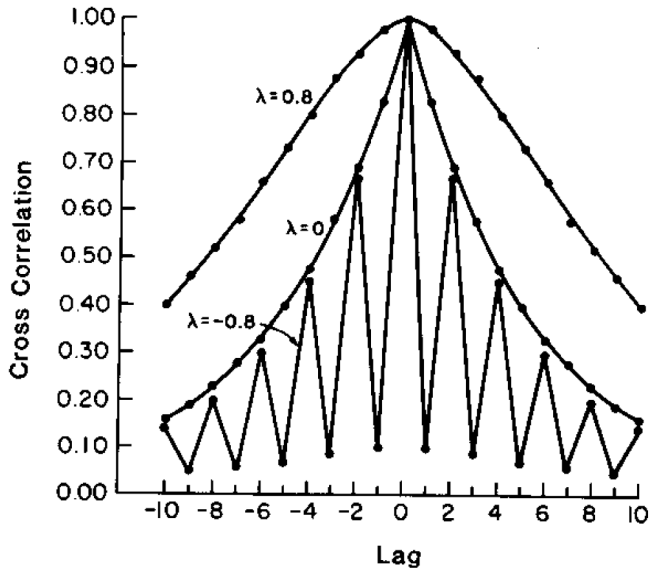
For a long series, like the financial data, if we want to predict the volume in the future we may need to only examine a short, recent period. A fixed number or lag (L) of say five or so days may suffice. Clearly a single series can then be divided into may such smaller bits – like the fecundity data we examined.

Thus, to predict $Y=v_t$, we will have a feature matrix, $X = \{X_1,..,X_L\}$, with this structure, $X_1 = \begin{pmatrix} v_{t-L} \\ r_{t-L} \\ z_{t-L} \end{pmatrix}$,..., $X_L = \begin{pmatrix} v_{t-1} \\ r_{t-1} \\ z_{t-1} \end{pmatrix}$.

Time Series

- For a single variable we can estimate the correlation between observations separated by 1, 2, …,*L* time units, e.g. Cor(v_t,v_{t-1}). This is called the autocorrelation. We can also estimate what is called the cross-correlation, Cor(v_t, r_{t-1}) for various lags (note, Cor(v_t, r_{t-1})≠Cor(r_t ,v_{t-1})). See Jenkins and Watts, 1968, *Spectral Analysis and its Applications*.
- If we are interested in understanding the time series we might develop a model of selection in a variable environment. Otherwise, a RNN might be useful for prediction.





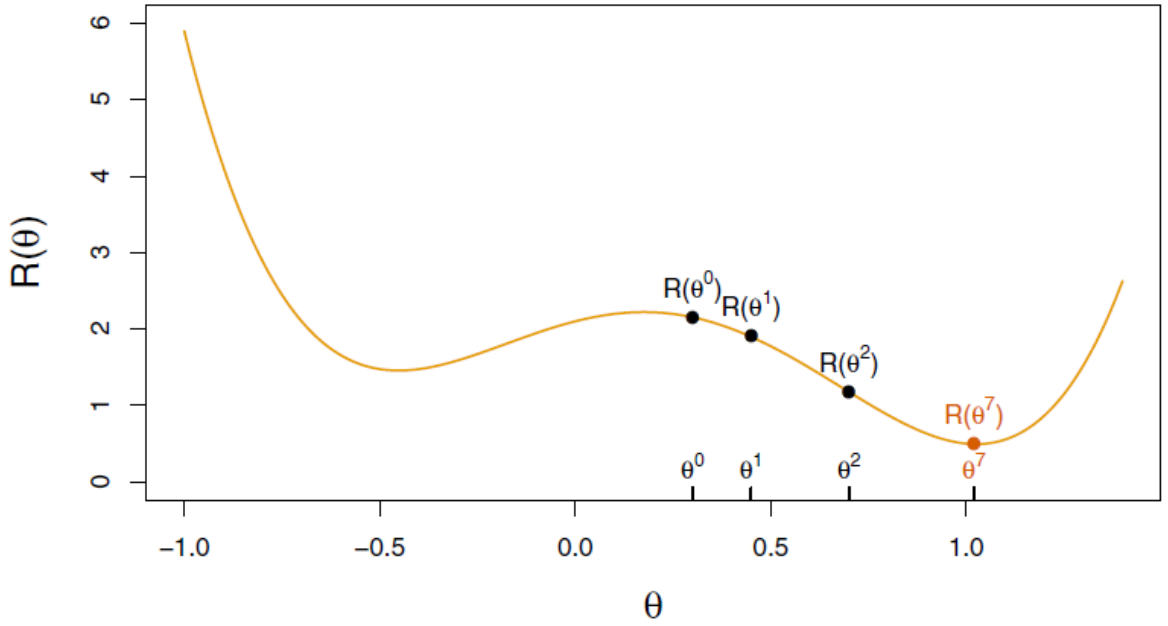
Mueller, L.D., L.G. Barr and F.J. Ayala, 1985. Natural selection versus random genetic drift: evidence from temporal variation in allele frequencies in nature. *Genetics* **111**: 517-554.

When to use deep learning

- If we want a model we can interpret then neural networks will be a poor choice and a linear model, GAM, or lasso might be a better choice.
- The neural networks will typically require very large training sets and this requirement alone may limit the use of neural networks.
- As we saw with the Hitters database, even when we can use neural networks they won't necessarily do better than simpler models.

Fitting a Neural Network

- For our simple one layer model the fitting process can be summarized as, $\underset{\{w_k\}_1^K, \beta}{minimize}\ \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2$.
- The difficulty come about because there are multiple minima in these non-linear, nonconvex functions.
- To overcome these problems and prevent overfitting there are two general strategies,
  (i) Slow learning, in conjunction with a technique called gradient descent.
  (ii) Regularization, usually the ridge or lasso.



Fitting a Neural Network

- Consider all the parameters in a single vector, θ. Then the minimization problem becomes, $R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$.
- Algorithm:
  1. Start with a guess θ^0 and t=0.
  2. Iterate until R(θ) fails to decrease.
     (a) Find a vector δ that satisfies $R(\theta^{t+1}) < R(\theta^t)$ where $\theta^{t+1} = \theta^t + \delta$.
     (b) Set $t \leftarrow t + 1$

Backpropagation

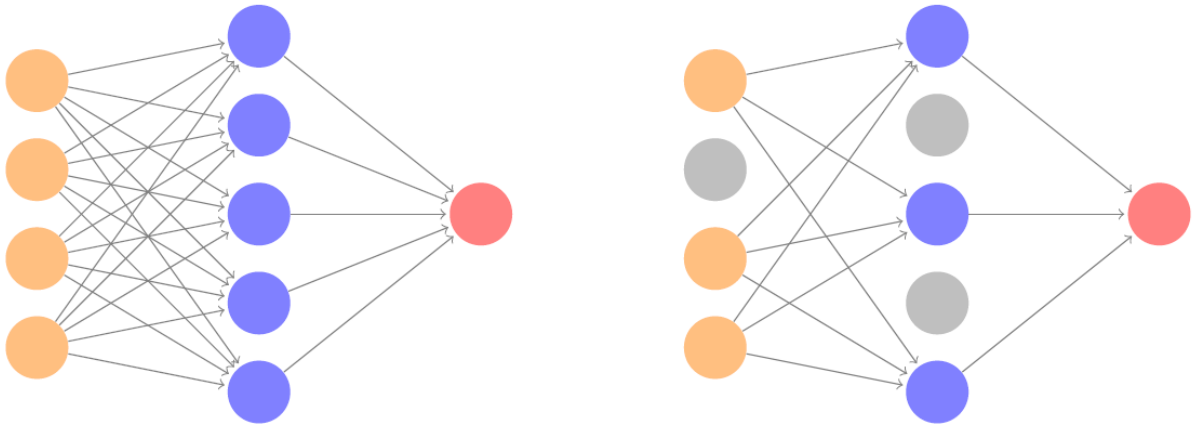
- To determine the direction of change for θ we calculate the gradient of the objective function, $\nabla R(\theta^m) = \frac{\partial R(\theta)}{\partial \theta}|_{\theta=\theta^m}$ where θ^m is the current value of θ.
- This derivative gives the direction of steepest increase in $R(\theta)$. To minimize $R(\theta)$ we go in the opposite direction and thus update θ as $\theta^{m+1} \leftarrow \theta^m - \rho \nabla R(\theta^m)$, ρ is the learning rate.
- The magnitude of ρ can be adjusted to insure $R(\theta^{t+1}) < R(\theta^t)$.
- The calculation of the gradient actually follows the chain rule for differentiation (see details on page 436).
- The partial derivatives of $R(\theta)$ with respect to w and β shows that a fraction of the residual, $y_i - f_\theta(x_i)$, affects each parameter which is referred to as backpropagation in the neural network literature.

Regularization and Stochastic Gradient Descent

- For a large database and large model the calculation of the gradient $\nabla R(\theta^m)$ can be time consuming.
- One way to compute this is to take a small sample of the database called a minibatch at each gradient step. This process is known as stochastic gradient descent (SGD).
- Regularization can be accomplished by adding a penalty function to the objective function, $\mathrm{R}(\theta,\lambda) = -\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log\left(f_m(x_i)\right) + \lambda\sum_j \theta_j^2$. The lasso can also be used.
- The progress of the fitting is followed through a number of epochs. An epoch is the number of gradient updates equal to the size of the training set divided by the size of the minibatch.

Dropout Learning

- Similar to a random forest a fraction, ∅, of the units in a layer, including the input, will be dropped. More precisely their output is set to zero.
- This dropout happens every time a training observation is processed.
- This prevents units nodes from becoming to “specialized” and is considered a form of regularization.



Network Tuning

- *The number of hidden layers and units per layer*. Many think the number if units per layer can be large and regularization techniques will prevent overfitting.
- *Regularization tuning parameters.* Adjust parameters like the dropout parameter and strength of the penalty function in a ridge or lasso estimator.
- *Details of the stochastic descent.* Adjust batch size, number of epochs, and data augmentation. An example of data augmentation is changing for the zoom level, rotation, horizontal and vertical flips of an image used in image recognition.

Interpolation and Double Descent

- Test MSE usually has a U shaped MSE function for test data. When the degrees of freedom of a polynomial equals the sample size there is a single least squares solution which goes through all points but its wild fluctuations produce a very high test MSE.
- If we the degrees of freedom exceed *n* there are an infinite number of solutions and under the right conditions the test MSE can actually decrease.

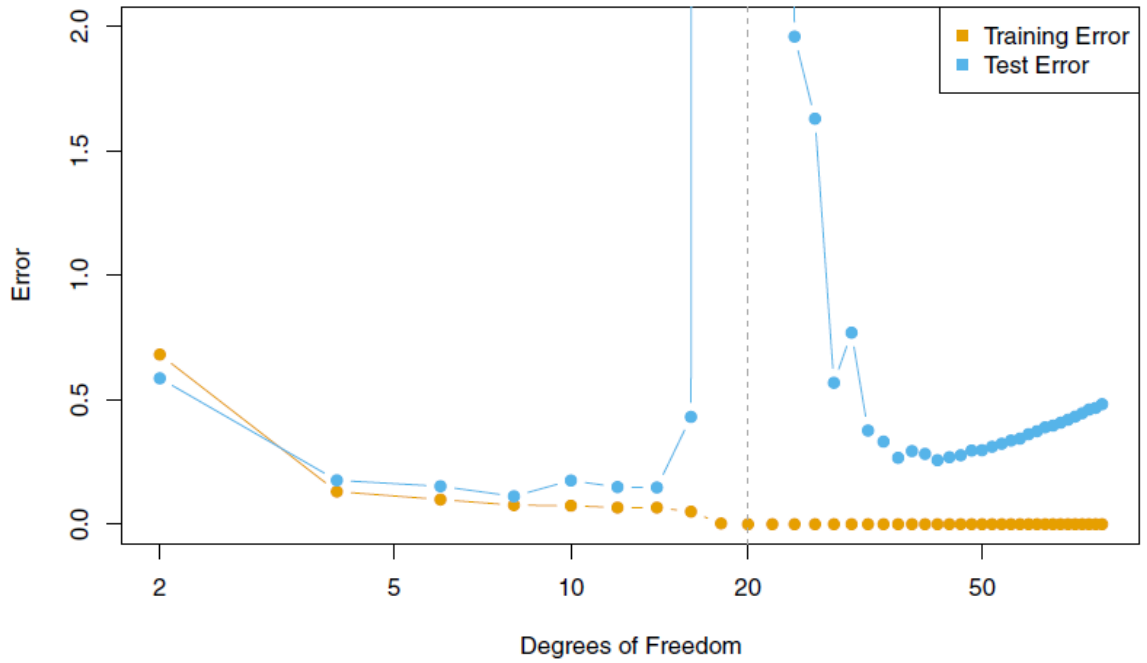


FIGURE 10.20. *Double descent phenomenon, illustrated using error plots for a one-dimensional natural spline example. The horizontal axis refers to the number of spline basis functions on the log scale. The training error hits zero when the degrees of freedom coincides with the sample size $n = 20$, 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.*

Interpolation and Double Descent

- When a spline fit with degrees of freedom (d) that exceed n, we can add the additional requirement among the many solutions that we seek on will minimize $\sum_{j=1}^{d} \hat{\beta}_j^2$.
- This results in a fit which is actually less wild than the interpolating polynomial.
- Similar phenomenon can happen in neural networks especially with data that has a high signal to noise ratio