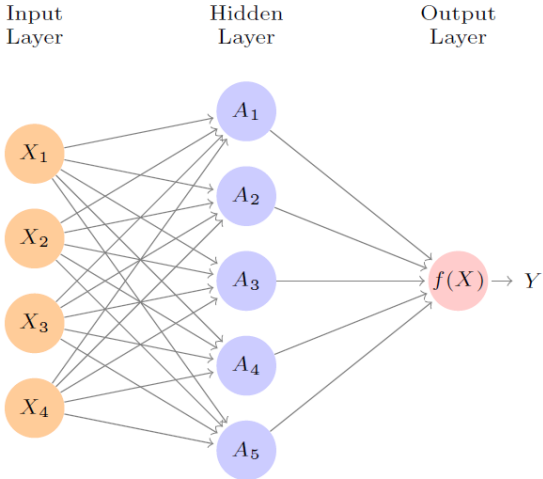
10.7 Fitting a Neural Network

Fitting Neural Networks

- Complex task, so we give just a brief overview for a single setting.
  - Ideas generalize to more complex networks.

Fitting Neural Networks



Consider the simple network we saw in Section 10.1

Fitting Neural Networks

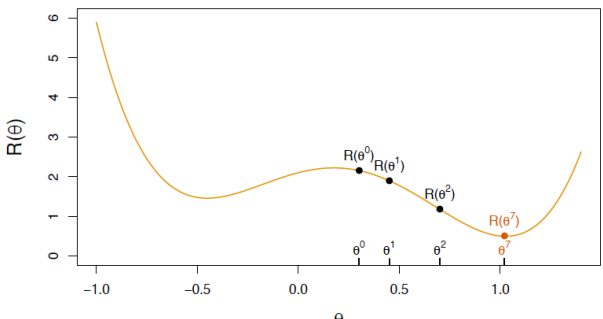
We fit the model by solving a nonlinear least squares problem.

$$\underset{\{w_k\}_1^K,\,\beta}{\text{minimize}}\ \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2,$$

$$f(x_i) = \beta_0 + \sum_{k=1}^{K}\beta_k g\Big(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}\Big).$$

Fitting Neural Networks

Because of the nested nature of the parameters and the symmetry of the hidden units, this is a challenging problem!



- The problem is non convex in the parameters, so there are multiple solutions.

Fitting Strategies

- *Slow learning*: the model is fit in a somewhat slow iterative fashion using **gradient descent**. The fitting process is stopped when overfitting is detected.
- *Regularization*: penalties are imposed on the parameters (usually lasso or ridge).

Fitting Neural Networks

If we represent all of the parameters in one long vector θ, we can rewrite the objective as

$$R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

where we make explicit the dependence of f on the parameters.

Gradient descent can then be conceptualized in a simple way.

Gradient Descent

1. Start with a guess θ^0 for all the parameters in θ and set $t = 0$.
2. Iterate until the objective $R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$ fails to decrease:
   a. Find a vector δ that reflects a small change in θ such that $\theta^{t+1} = \theta^t + \delta$ reduces the objective.
   b. Set $t \leftarrow t + 1$

Gradient Descent

- As long as each step goes downhill, we will eventually get to the bottom of the hill.
- However, we're not guaranteed that our path will take us to the *global* minimum.
- In general, we hope to end up at a (good enough) local minimum.

Backpropagation

How do we might the directions to move θ so as to decrease $R(\theta)$?

- The *gradient* of $R(\theta)$ evaluated at some current value $\theta = \theta^m$ is the vector of partial derivatives at that point

$$\nabla R(\theta^m) = \left.\frac{\delta R(\theta)}{\delta \theta}\right|_{\theta=\theta^m}$$

- This gives the direction in θ-space in which $R(\theta)$ *increases* most rapidly.

Backpropagation

- Gradient descent moves θ a bit in the opposite direction

$$\theta^{m+1} \leftarrow \theta^m - \rho \nabla R(\theta^m)$$

  - For a small enough value of the *learning rate*, ρ, this step will decrease the objective.
  - If the gradient value is 0, we may have arrived at a minimum of the objective.

Backpropagation

$R(\theta) = \sum_{i=1}^{n} R_i(\theta)$ is a sum, so the gradient is a sum of gradients.

$$R_i(\theta) = \tfrac{1}{2}(y_i - f_\theta(x_i))^2 = \tfrac{1}{2}\Big(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k g\big(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\big)\Big)^2$$

For ease of notation, let $z_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}$.

Backpropagation uses the *chain rule for differentiation*:

$$\begin{aligned}
\frac{\partial R_i(\theta)}{\partial \beta_k} &= \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial \beta_k} \\
&= -(y_i - f_\theta(x_i)) \cdot g(z_{ik}). \\
\frac{\partial R_i(\theta)}{\partial w_{kj}} &= \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}} \\
&= -(y_i - f_\theta(x_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}.
\end{aligned}$$

Backpropagation

- Each expression contains the residual $y_i - f_\theta(x_i)$
- A fraction of that residual gets attributed to each of the hidden units.
- In fact, the act of differentiation assigns a fraction of the residual to each of the parameters via the chain rule.
  - This process is what's known as *backpropagation*.

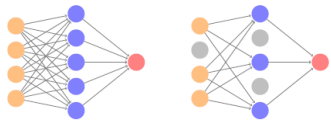
Tricks of the Trade

- *Slow learning*. Gradient descent is slow, and a small learning rate ρ slows it even further.
  - With *early stopping*, this is a form of regularization.
- *Stochastic gradient descent*. Rather than compute the gradient using all of the data, we use a small "minibatch" drawn at random at each step.
  - E.g., for the `MNIST` data, with $n = 60K$, we use minibatches of 128 observations.

Tricks of the Trade

- An *epoch* is a count of iterations and amounts to the number of minibatch updates such that n samples in total have been processed.
  - That is, $60K/128 \approx 469$ for `MNIST`
- *Regularization*. Ridge and lasso can be used to shrink the weights at each layer. Two other popular forms of regularization are *dropout* and *augmentation*.

Dropout Learning



- At each SGD update, randomly remove units with probability ϕ, and scale up the weights of those retained by $1/(1-\phi)$ to compensate.
- In simple scenarios like linear regression, a version of this process can be shown to be equivalent to ridge regularization.
- The other units *stand in* for those temporarily removed, and their weights are drawn closer together.
- Similar to randomly omitting variables with growing trees in random forests.

Network Tuning

Even on simple networks, we have a number of choices to make:

- The number of hidden layers,
- the number of units per layer,
  - Modern thinking is that the number of units per layer can be large, with overfitting controlled via regularization.
- regularization tuning parameters,
  - These include the dropout rate ϕ and the strength of λ of lasso and ridge, and are typically set separately at each layer.
- details of stochastic gradient descent
  - Includes batch size, number of epochs, and any details of data augmentation,

These choices can make a big difference and tinkering with them can be very tedious!