Nonlinear models for regression

Michela Mulas

A quick recap

During the last lectures, we did...

- **Linear models for regression**.
  Simple least squares models
  Multiple least squares models
  Penalized regression models
  Principal components regression
  Partial least squares regression

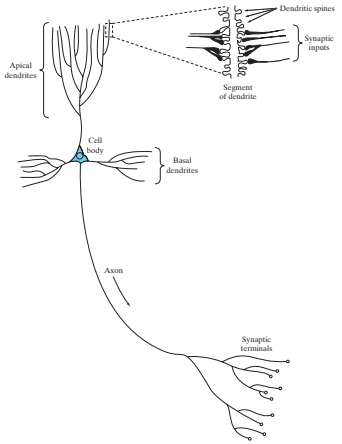
A quick recap

During the last lectures, we did...

- **Nonlinear models for regression**.
  Introduce neural networks starting with some naive neurobiology.

  Neural networks are powerful nonlinear regression techniques inspired by theories about how the brain works.

  Functional units of a biological neuron are:
  - Cell body
  - Dendrites
  - Axon
  - Synaptic gaps
  - Stimulus
  - ...



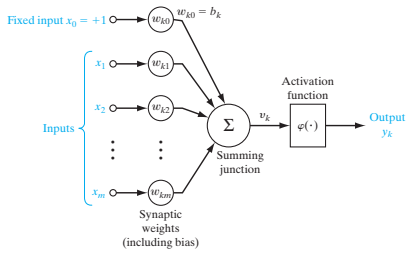
A quick recap

During the last lectures, we did...

- **Nonlinear models for regression**.
  Introduce the simplest artificial neural network: the neuron.

  The problem we addressed is how to design a multiple-input-single-output model of the unknown dynamic system by building it around a single linear neuron.

  The neural model operates under the influence of an algorithm that controls necessary adjustments to the synaptic weights of the neuron.



A quick recap

During the last lectures, we did...

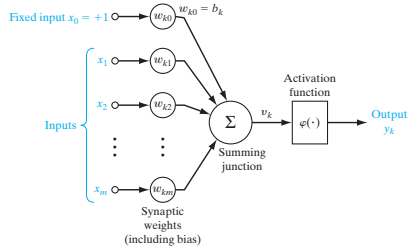
- **Nonlinear models for regression**.

  Introduce the simplest artificial neural network: the neuron.

  We described the neuron k as:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j$$

$$y_k = \varphi(v_k)$$



Today's goal

Today, we going to ...

- Nonlinear models for regression.
- **Neural networks**:

  The learning problem.

  The back-propagation algorithm.

Reference

Simon Haykin. *Neural Networks: A Comprehensive Foundation*, Pearson (2008, 3rd edition).

Unconstrained optimization

Learning consists of adjusting weight and threshold values until a certain criterion (or several criteria) is (are) satisfied $\rightsquigarrow$ It is an optimization problem.

Consider a cost function $E(\mathbf{w})$

- It is a continuous differentiable function of some unknown weight vector $\mathbf{w}$.
- The function $E(\mathbf{w})$ maps the elements of $\mathbf{w}$ into real numbers.
- It is a measure of how to choose the weight (parameter) vector $\mathbf{w}$ of an adaptive-filtering algorithm so that it behaves in an optimum manner.
- We want to find an optimal solution $\mathbf{w}^*$ that satisfies the condition:

$$E(\mathbf{w}^*) \leq E(\mathbf{w})$$

Minimize the cost function $E(\mathbf{w})$ with respect to the weight vector w.

Unconstrained optimization

The necessary condition for optimality is:

$$\nabla E(\mathbf{w}^*) = 0$$

- ∇ is the **gradient operator**

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_1}, \ldots, \frac{\partial}{\partial w_M}\right]^T$$

- $\nabla E(\mathbf{w})$ is the **gradient vector** of the cost function:

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_M}\right]^T$$

Unconstrained optimization

A class of unconstrained-optimization algorithms that is particularly well suited for the design of adaptive filters is based on the idea of local iterative descent.

- Starting with an initial guess denoted by $\mathbf{w}(0)$, generate a sequence of weight vectors $\mathbf{w}(1), \mathbf{w}(2), \ldots$, such that the cost function $E(\mathbf{w})$ is reduced at each iteration of the algorithm:

$$E(\mathbf{w}(n+1)) < E(\mathbf{w}(n))$$

$\mathbf{w}(n)$ is the old value of the weights and $\mathbf{w}(n+1)$ is its update value.

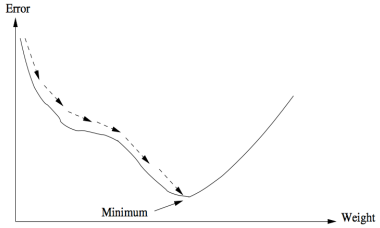
Hopefully the algorithm will eventually converge.

Unconstrained optimization

Steepest descent method: the successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, that is, in a direction opposite to the gradient vector $\nabla E(\mathbf{w})$.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta \underbrace{\nabla E(\mathbf{w})(n)}_{\mathbf{g}}$$

- η is a positive constant: **learning rate**.
- $\mathbf{g}$ is the gradient vector evaluated at the point $\mathbf{w}(n)$.



From iteration n to $n+1$, the algorithm applies the correction:

$$\Delta\mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n) = -\eta\mathbf{g}(n)$$

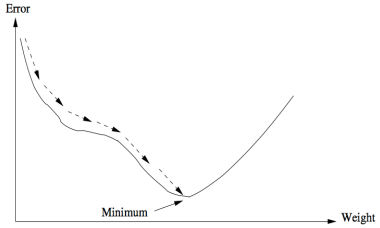
Unconstrained optimization

Steepest descent method: the successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, that is, in a direction opposite to the gradient vector $\nabla E(\mathbf{w})$.

The steepest-descent algorithm satisfies the condition $E(\mathbf{w}(n+1)) < E(\mathbf{w}(n))$ for iterative descent.

In fact, substituting the correction $\Delta\mathbf{w}(n)$ into a first-order Taylor series expansion around $\mathbf{w}(n)$ to approximate $E(\mathbf{w}(n+1))$ gives

$$\begin{aligned} E(\mathbf{w}(n+1)) &\approx E(\mathbf{w}(n)) + \mathbf{g})^T(n)\mathbf{g}(n) \\ &= E((w)(n)) - \eta||\mathbf{g}||^2 \end{aligned}$$



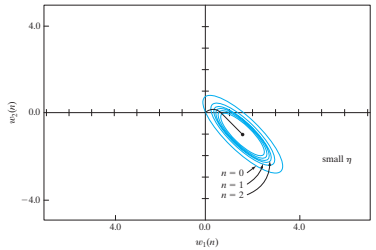
For a positive learning-rate parameter η, the cost function is decreased as the algorithm progresses from one iteration to the next.

Unconstrained optimization

Steepest descent method: the successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, that is, in a direction opposite to the gradient vector $\nabla E(\mathbf{w})$.

The steepest-descent converges to the optimal solution $\mathbf{w}^*$ slowly.

- η small: the transient response of the algorithm is overdamped, in that the trajectory traced by $\mathbf{w}(n)$ follows a smooth path in the W-plane

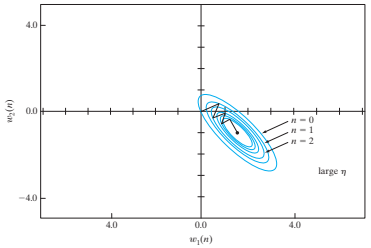


Unconstrained optimization

Steepest descent method: the successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, that is, in a direction opposite to the gradient vector $\nabla E(\mathbf{w})$.

The steepest-descent converges to the optimal solution $\mathbf{w}^*$ slowly.

- η large: the transient response of the algorithm is underdamped, in that the trajectory of $\mathbf{w}(n)$ follows a zigzagging (oscillatory) path.
- η exceeds a certain critical value, the algorithm becomes unstable (i.e., it diverges).



Unconstrained optimization

Newton's method minimizes the quadratic approximation of the cost function $E(\mathbf{w})$ around the current point $\mathbf{w}(n)$.

- This minimization is performed at each iteration of the algorithm.
- Using a second-order Taylor series expansion of the cost function around the point $\mathbf{w}(n)$, we have:

$$\Delta E(\mathbf{w}(n)) = E(\mathbf{w}(n+1)) - E(\mathbf{w}(n)) \approx \mathbf{g}^T \Delta \mathbf{w}(n) + \frac{1}{2}\Delta \mathbf{w}^T(n)\mathbf{H}(n)\Delta \mathbf{w}(n)$$

The matrix $\mathbf{H}$ is the m-by-m Hessian of $E(\mathbf{w})$:

$$\mathbf{H} = \nabla^2 E(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_M} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_M} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 E}{\partial w_M \partial w_1} & \frac{\partial^2 E}{\partial w_M \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_M^2} \end{bmatrix}$$

Unconstrained optimization

Newton's method minimizes the quadratic approximation of the cost function $E(\mathbf{w})$ around the current point $\mathbf{w}(n)$.

- This minimization is performed at each iteration of the algorithm.
- Using a second-order Taylor series expansion of the cost function around the point $\mathbf{w}(n)$, we have:

$$\Delta E(\mathbf{w}(n)) = E(\mathbf{w}(n+1)) - E(\mathbf{w}(n)) \approx \mathbf{g}^T \Delta \mathbf{w}(n) + \frac{1}{2}\Delta \mathbf{w}^T(n)\mathbf{H}(n)\Delta \mathbf{w}(n)$$

Differenziating with respect to $\Delta\mathbf{w}$, the change of $\Delta E(\mathbf{w})$ is minimized when:

$$\mathbf{g}(n) + \mathbf{H}(n)\Delta \mathbf{w}(n) = \mathbf{0}$$

Solving for $\Delta\mathbf{w}(n)$ and substituting, we have:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta \mathbf{w}(n) = \mathbf{w}(n) - \mathbf{H}^{-1}(n)\mathbf{g}(n)$$

- Newton's method converges quickly asymptotically and does not exhibit the zigzagging behaviour.
- $\mathbf{H}(m)$ has to be a positive definite matrix for all n.

Unconstrained optimization

Gauss-Newton method is applicable to a cost function that is expressed as:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

- The error signal $e(i)$ is a function of the adjustable weight vector $\mathbf{w}$.
- Given an operating point $\mathbf{w}(n)$, we linearize the dependence of $e(i)$ on $\mathbf{w}$:

$$e'(i,\mathbf{w}) = e(i) + \underbrace{\left[\frac{\partial e(i)}{\partial \mathbf{w}}\right]^T_{\mathbf{w}=\mathbf{w}(n)}}_{\text{It gives the } \textbf{Jacobian matrix}} (\mathbf{w} - \mathbf{w}(n)), \quad i = 1,2,\ldots,n$$

- By using matrix notation:

$$\mathbf{e}'(n,\mathbf{w}) = \mathbf{e}(n) + \mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n))$$

$\mathbf{e}$ is the error vector

$$\mathbf{e}(n) = [e(1), e(2), \ldots, e(n)]^T$$

Unconstrained optimization

Gauss-Newton method is applicable to a cost function that is expressed as:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

- $\mathbf{J}(n)$ is the n-by-m Jacobian matrix of $\mathbf{e}(n)$:

$$\mathbf{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \cdots & \cdots & \cdots & \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{bmatrix}$$

Unconstrained optimization

Gauss-Newton method is applicable to a cost function that is expressed as:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

- The Gauss-Newton method results as:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \Big(\underbrace{\mathbf{J}^T(n)\mathbf{J}(n)}_{\text{It must be nonsingular}} \Big)^{-1} \mathbf{J}(n)\mathbf{e}(n)$$

- To guard against the possibility the $\mathbf{J}(n)$ is rank deficient (then singular), we add the diagonal matrix $\delta\mathbf{I}$ to $\mathbf{J}^T(n)\mathbf{J}(n)$.

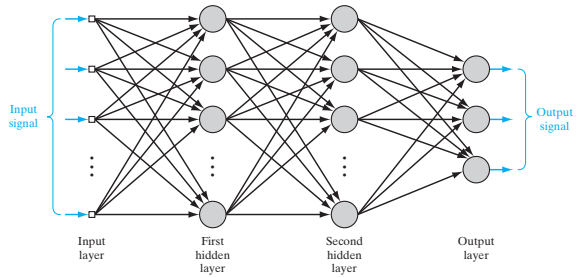
  The parameter δ is a small positive constant chosen to ensure that:

$$\mathbf{J}^T(n)\mathbf{J}(n) + \delta\mathbf{I}: \quad \text{positive definite } \forall n$$

Back-propagation algorithms

Basic features of a typical neural network (multilayer perceptron) are:

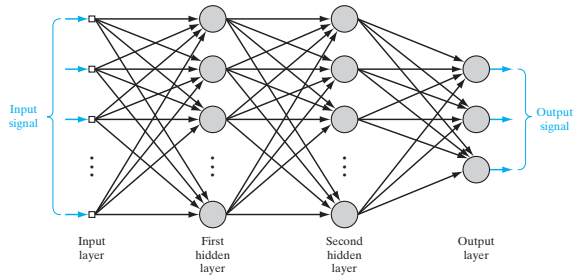
- The model of each neuron includes a nonlinear and differentiable activation function.
- The network contains one or more layers that are hidden from both the input and output nodes.
- The network exhibits a high degree of connectivity, the extent of which is determined by synaptic weights of the network.



Back-propagation algorithms

The same features make somehow difficult to understand how the networks work:

- The presence of a distributed form of nonlinearity and the high connectivity of the network make the theoretical analysis of a multilayer perceptron difficult to undertake.
- The hidden neurons makes the learning process harder to visualize.
- The learning process is made more difficult because the search has to be conducted in a much larger space of possible functions.



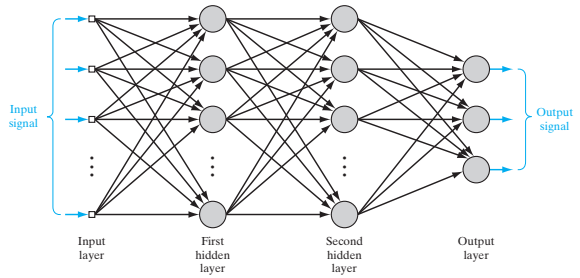
Back-propagation algorithms

A popular method for the training the networks is the back-propagation algorithm.

The training proceeds in two phases:

1. In the **forward phase**, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output.

   In this phase, changes are confined to the activation potentials and outputs of the neurons in the network.



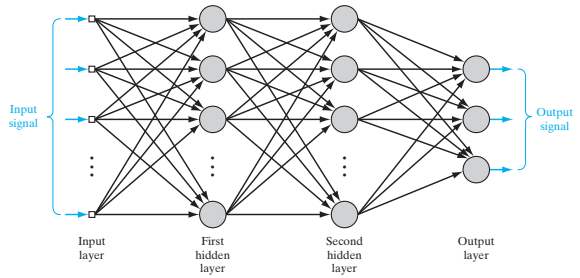
Back-propagation algorithms

A popular method for the training the networks is the back-propagation algorithm.

The training proceeds in two phases:

2. In the **backward phase**, an error signal is produced by comparing the output of the network with a desired response.

   The resulting error signal is propagated through the network, again layer by layer, but this time the propagation is performed in the backward direction.



Back-propagation algorithms

A popular method for the training the networks is the back-propagation algorithm.

The training proceeds in two phases:

2. In this second phase, successive adjustments are made to the synaptic weights of the network.

   Calculation of the adjustments for the output layer is straightforward, but it is much more challenging for the hidden layers.