Pattern Recognition 2015
Neural Networks (1)

Ad Feelders

Universiteit Utrecht

Neural Networks

The term *neural network* has its origins in attempts to find mathematical representations of information processing in biological systems.

From the perspective of practical applications of pattern recognition, however, biological realism would impose entirely unnecessary constraints.

The story so far

Linear regression and classification

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right) \tag{5.1}$$

where the ϕ_j are (non-linear) transformations of the input variables, that do not depend on any unknown parameters.

$f(\cdot)$: identity for linear regression, and logistic sigmoid for logistic regression.

Feed-forward Network Functions

Let basis functions $\phi_j(\mathbf{x})$ depend on unknown parameters that are adjusted to the training data.

The basis functions follow the same form as (5.1).

Basic Neural Network Model

Construct M linear combinations of the input variables $x_1, \ldots, x_D$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{5.2}$$

for $j = 1, \ldots, M$.

Each a_j is transformed using a non-linear *activation function* $h(\cdot)$ to give

$$z_j = h(a_j). \tag{5.3}$$

Basic Neural Network Model

The z_j values are again linearly combined to give the output unit activations

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{5.4}$$

for $k = 1, \ldots, K$.

$$y_k = a_k \qquad \text{(regression)}$$
$$y_k = \sigma(a_k) \qquad \text{(binary classification)}$$

Basic Neural Network Model

Output as function of input (logistic output)

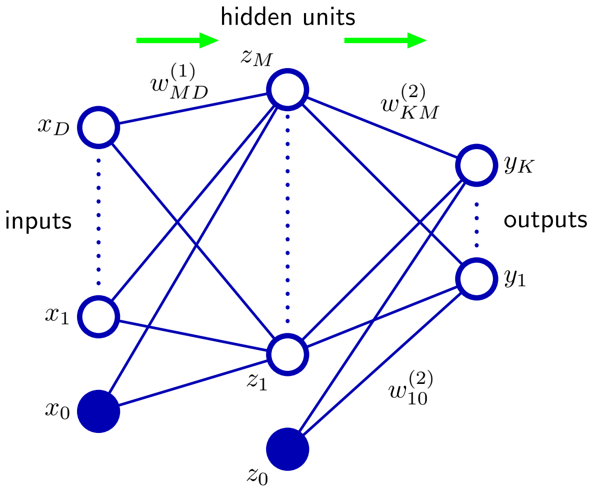
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \underbrace{\left(\sum_{j=1}^{M} w_{kj}^{(2)} h \underbrace{\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right)}_{a_j} + w_{k0}^{(2)} \right)}_{a_k}, \qquad (5.7)$$

where

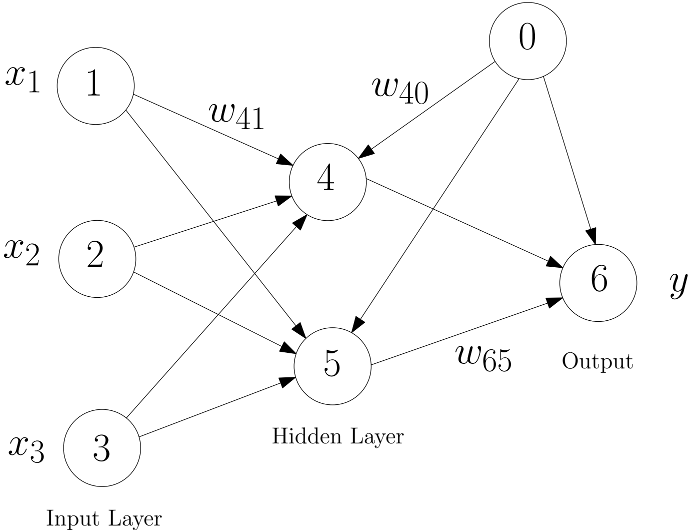
$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

is the logistic sigmoid activation function.

Network Diagram



Example feed-forward neural network



Example feed-forward neural network

This network corresponds to the following non-linear regression function (linear output, sigmoid activation function in hidden units)

$$\begin{aligned} y(\mathbf{x}, \mathbf{w}) &= w_{60} + w_{64} \left(\frac{e^{w_{40}+w_{41}x_1+w_{42}x_2+w_{43}x_3}}{1 + e^{w_{40}+w_{41}x_1+w_{42}x_2+w_{43}x_3}} \right) \\ &+ w_{65} \left(\frac{e^{w_{50}+w_{51}x_1+w_{52}x_2+w_{53}x_3}}{1 + e^{w_{50}+w_{51}x_1+w_{52}x_2+w_{53}x_3}} \right) \end{aligned}$$

Error Function

- Regression: squared error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \tag{5.14}$$

- Binary classification: cross-entropy (negative loglikelihood)

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\} \tag{5.21}$$

where y_n denotes $y(\mathbf{x}_n, \mathbf{w})$.

For neural networks, the error function is a complex non-linear function of the weights, requiring numerical optimization to find optimal values for them.

Parameter Optimization (single parameter)

Suppose the error function only depends on a single weight w. From calculus we know that a necessary condition for a minimum is:

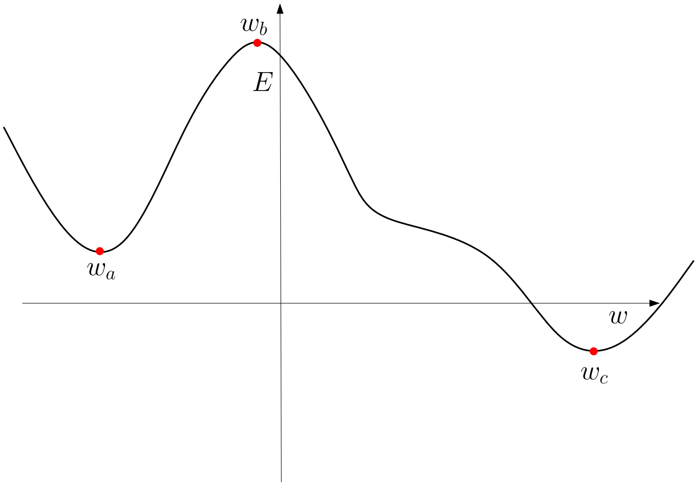
$$\frac{dE}{dw} = 0 \tag{1}$$

This condition is not sufficient, since maxima and points of inflection also satisfy equation (1). Together with the second-order condition:

$$\frac{d^2E}{dw^2} > 0, \tag{2}$$

we have a sufficient condition for a local minimum.

Error function $E(w)$



Parameter Optimization: Example

Regression through the origin:

$$y(x) = wx$$

$D = \{(x_1, t_1), (x_2, t_2)\} = \{(2, 5), (1, 3)\}.$

$$E(w) = (5 - 2w)^2 + (3 - w)^2$$

The first derivative is

$$\frac{dE}{dw} = -4(5 - 2w) - 2(3 - w) = 10w - 26$$

Equate to zero and solve for w:

$$10w - 26 = 0 \Rightarrow w = 2.6$$

Parameter Optimization: Example

To determine whether we have found a minimum, compute

$$\frac{d^2E}{dw^2} = \frac{d10w - 26}{dw} = 10 > 0$$

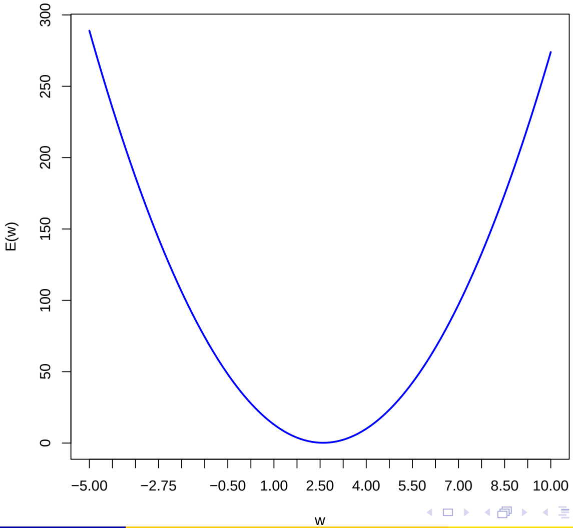
In general

$$E(w) = \sum_{n=1}^{N} (t_n - wx_n)^2$$

$$\frac{dE}{dw} = -2\sum_{n=1}^{N}(t_n - wx_n)x_n = -2\sum_{n=1}^{N} t_n x_n + 2w\sum_{n=1}^{N} x_n^2$$

$$\frac{d^2E}{dw^2} = 2\sum_{n=1}^{N} x_n^2 > 0$$

Error function $E(w)$ for regression through the origin example



Parameter Optimization (multiple parameters)

Usually the error function depends on multiple weights $w_1, \ldots, w_m$. Analogous to the single-parameter case a necessary condition for a minimum is:

$$\frac{\partial E}{\partial w_j} = 0 \qquad j = 1, \ldots, m \tag{3}$$

Again this condition is not sufficient, since maxima and saddle points also satisfy (3). For the second order condition, define the Hessian matrix $\mathbf{H}$, with

$$\mathbf{H}_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$$

Together with the second-order condition that $\mathbf{H}$ is positive definite, i.e.

$$\mathbf{z}^\top \mathbf{H} \mathbf{z} > 0, \qquad \text{for all } \mathbf{z} \neq \mathbf{0} \tag{4}$$

we have a sufficient condition for a local minimum.

Parameter Optimization

The smallest value of $E(\mathbf{w})$ will occur at a point in weight space where the gradient vanishes, so that

$$\nabla E(\mathbf{w}) = \mathbf{0}, \tag{5.26}$$

where the gradient is the vector of partial derivatives

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \vdots \\ \frac{\partial E}{\partial w_m} \end{bmatrix}$$

Parameter Optimization

Consider a second-order Taylor expansion of $E(\mathbf{w})$ around $\widehat{\mathbf{w}}$

$$E(\mathbf{w}) \approx E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^\top \mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^\top \mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}}) \tag{5.28}$$

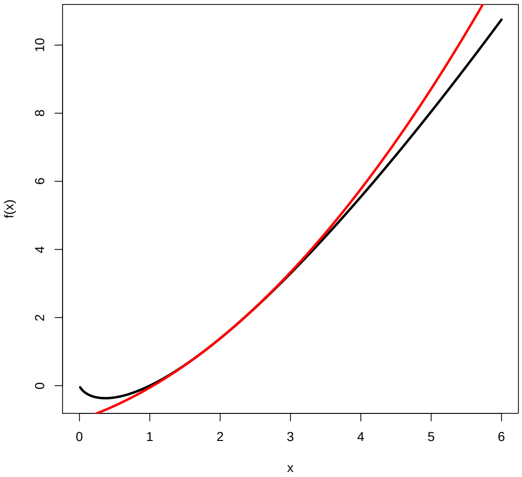
where

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\widehat{\mathbf{w}}} \tag{5.29}$$

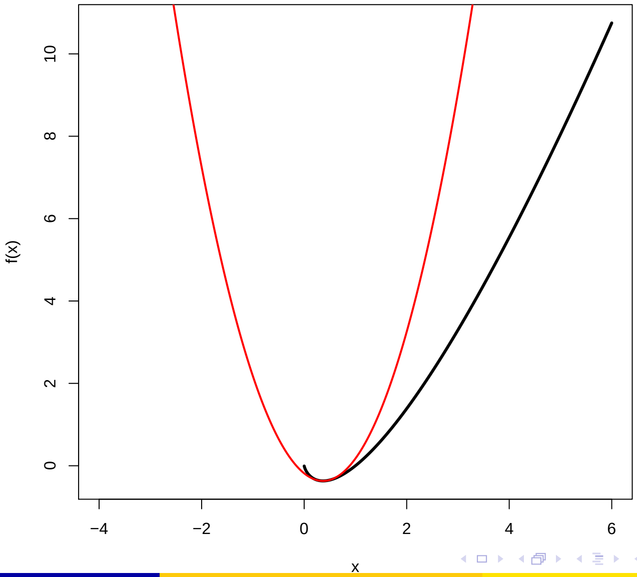
and

$$\mathbf{H}_{ij} \equiv \left.\frac{\partial^2 E}{\partial w_i \partial w_j}\right|_{\mathbf{w}=\widehat{\mathbf{w}}} \tag{5.30}$$

Quadratic approximation of $x \ln x$ at $\hat{x} = 2$.



Quadratic approximation of $x \ln x$ at $\hat{x} = e^{-1}$.



Local Quadratic Approximation

Let $\mathbf{w}^\star$ be a stationary point of the error function, i.e. $\nabla E = \mathbf{0}$ at $\mathbf{w}^\star$.
Then (5.28) becomes

$$E(\mathbf{w}) \approx E(\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^\star)^\top \mathbf{H}(\mathbf{w} - \mathbf{w}^\star) \qquad (5.32)$$

Hence, we have

$$E(\mathbf{w}) - E(\mathbf{w}^\star) = \frac{1}{2}(\mathbf{w} - \mathbf{w}^\star)^\top \mathbf{H}(\mathbf{w} - \mathbf{w}^\star)$$

Now, suppose

$$(\mathbf{w} - \mathbf{w}^\star)^\top \mathbf{H}(\mathbf{w} - \mathbf{w}^\star) > 0 \qquad \text{for all } (\mathbf{w} - \mathbf{w}^\star) \neq \mathbf{0}, \qquad (5.37)$$

that is, $\mathbf{H}$ is positive definite.

Then we have $E(\mathbf{w}) - E(\mathbf{w}^\star) > 0$, that is, $E(\mathbf{w}) > E(\mathbf{w}^\star)$, so $\mathbf{w}^\star$ is a local minimum.

Numeric Optimization

So, to find a (local) minimum of the error function, we

- find a value $\mathbf{w}^\star$ with $\nabla E = \mathbf{0}$ at $\mathbf{w}^\star$.
- verify that $\mathbf{H}$ is positive definite at this point.

No hope of finding an analytical solution to (5.26), so we have to resort to iterative numerical procedures.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{5.27}$$

producing a sequence of iterates

$$\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots$$

that converges to a local minimum of the error function.

Gradient Descent

The gradient points in the direction of the *steepest ascent* of the function.

The *gradient-descent algorithm* is:

- *choose* an initial value $\mathbf{w}^{(0)}$;
- determine the gradient $\nabla E(\mathbf{w}^{(0)})$ of $E(\mathbf{w})$ at $\mathbf{w}^{(0)}$ and *update*

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \eta \nabla E(\mathbf{w}^{(0)})$$

- Repeat the previous step until

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

  and *check* if a *(local) minimum* has been reached.

$\eta > 0$ is the *step size* (or *learning rate* in NN literature).

On the step size

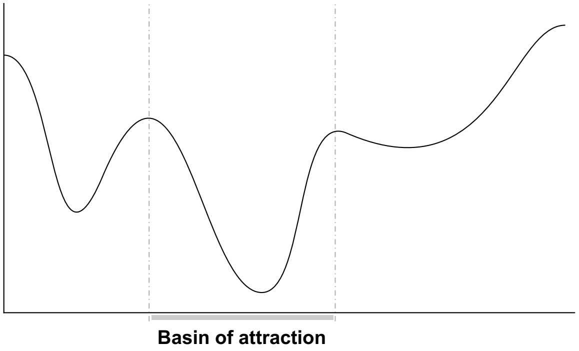
Choosing an appropriate *step size* (learning rate) for the gradient descent algorithm is *crucial* for its performance:

- if the step size is *too small*, then the algorithm may become *prohibitively slow*;
- if the step size is *too large*, then the algorithm may *overshoot* the minimum and may not even converge.

Taking a fixed step size is a very crude approach. More sophisticated approaches are possible (e.g. line search).

The basin of attraction

The gradient descent algorithm converges to a *local minimum*, but not necessarily to a global one:



To increase the probability of getting into the *basin of attraction*, the basic algorithm may be randomly *restarted* a number of times.

Example of gradient descent

Consider the function $E(w_1, w_2)$ with

$$E(w_1, w_2) = 3(w_1 + 1)^2 + (w_2 + 4)^2$$

The gradient $\nabla E(w_1, w_2) = [6w_1 + 6, 2w_2 + 8]^\top$ reveals the *stationary point* $(-1, -4, 0)$. The *Hessian matrix* equals

$$\mathbf{H} = \begin{bmatrix} 6 & 0 \\ 0 & 2 \end{bmatrix}$$

for which we find that

- the first principal minor equals $\mathbf{H}_{11} = 6$;
- the second principal minor equals $|\mathbf{H}| = 6 \cdot 2 - 0 \cdot 0 = 12$.

Since $\mathbf{H}$ *positive definite*, we conclude that $(-1, -4, 0)$ is a *local minimum* of the function.

The example continued

Consider again the function $E(w_1, w_2)$ with

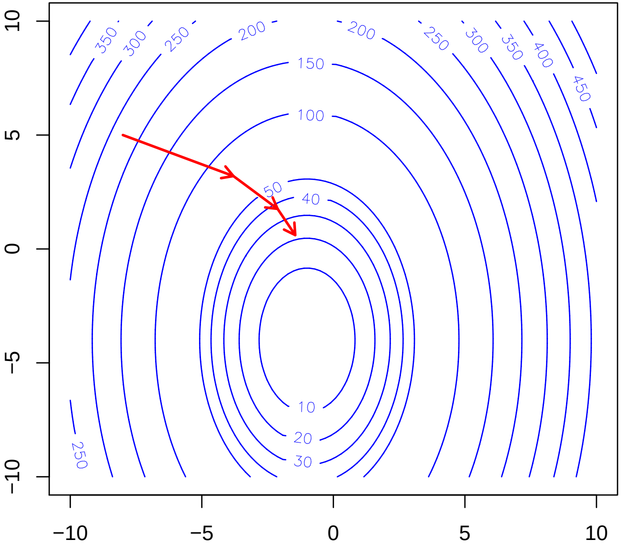
$$E(w_1, w_2) = 3(w_1 + 1)^2 + (w_2 + 4)^2$$

Suppose that the *gradient descent algorithm* is applied with *step size* $\frac{1}{10}$:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \frac{1}{10} \cdot \nabla E(\mathbf{w}^{(\tau)})$$

- at the initial input value $[w_1, w_2] = [-8, 5]$, the gradient equals $\nabla E(-8, 5) = [-42, 18]^\top$;
- the input value $[w_1, w_2]$ is updated to $[w_1, w_2] = [-8 + \frac{1}{10} \cdot 42, 5 - \frac{1}{10} \cdot 18] = [-3.8, 3.2]$;
- in the next step, we find that $\nabla E(-3.8, 3.2) = [-16.8, 14.4]^\top$ and $[w_1, w_2]$ is updated to $[w_1, w_2] = [-2.12, 1.76]$.

The example continued



350
300
250
200
200
250
300
350
400
450
150
100
50
40
10
20
30
250
10
5
0
−5
−10
−10
−5
0
5
10