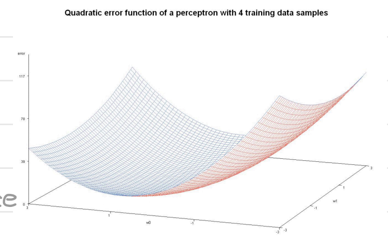
Non-linear Programming - Neural Networks



Intuition

input vector → weighted sum → output ("linear unit") OR activation fn → output } above cut-off ? 1 : 0

perceptron training → optimal weights → min error in hypothesis space

Section 1: Training single-layer perceptron: gradient descent

Section 2: Training multi-layer perceptron: back-propagation

Gradient descent

Initialisation step: Initialise weight vector (w) where each w_j is a small value.

Optimisation step: Until termination conditions are met, optimise w.

E.g. #ID (d)	Characteristic 1	Characteristic 2	Outcome (t_d)
1	0.2	0.9	1
2	0.1	0.1	0
3	0.2	0.4	0
4	0.2	0.5	0
5	0.4	0.5	1
6	0.3	0.8	1

bias / distortion

$w_0 = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$, learning rate $\boxed{\eta = 0.1}$ → must be a small positive value. If negative, algorithm performs gradient ascend

Step 1: Create table of predictions & errors

d	x			t_d	prediction score, $O_d = w^T x_d$	squared error, $(t_d - O_d)^2$
		Characteristic 1	Characteristic 2			
1	1	0.2	0.9	1	$(-1)(1) + (1)(0.2) + (1)(0.9) = 0.10$	$(1 - 0.10)^2 = 0.81$
2	1	0.1	0.1	0	-0.80	0.64
3	1	0.2	0.4	0	-0.40	0.16
4	1	0.2	0.5	0	-0.30	0.09
5	1	0.4	0.5	1	-0.10	1.21
6	1	0.3	0.8	1	0.10	0.81

Step 1.5: If required, compute "sum of square error"

$$\frac{1}{2}\Sigma(t_d - O_d)^2 = \frac{1}{2}(0.81 + 0.64 + 0.16 + 0.09 + 1.21 + 0.81) = 1.86$$

Step 2: Compute weight correction vector

$$\Delta w = \eta \Sigma_d (t_d - O_d) x_d = 0.1\left[(1-0.10)\begin{pmatrix} 1 \\ 0.2 \\ 0.9 \end{pmatrix} + \begin{pmatrix} 0.8 \\ 0.08 \\ 0.08 \end{pmatrix} + \begin{pmatrix} 0.4 \\ 0.08 \\ 0.16 \end{pmatrix} + \begin{pmatrix} 0.3 \\ 0.06 \\ 0.15 \end{pmatrix} + \begin{pmatrix} 1.1 \\ 0.44 \\ 0.55 \end{pmatrix} + \begin{pmatrix} 0.9 \\ 0.27 \\ 0.72 \end{pmatrix}\right] = \begin{pmatrix} 0.440 \\ 0.111 \\ 0.247 \end{pmatrix}$$

Step 3: Compute new weights

$$w_{new} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0.440 \\ 0.111 \\ 0.247 \end{pmatrix} = \begin{pmatrix} -0.560 \\ 1.111 \\ 1.247 \end{pmatrix}$$

Step 3.5: If required, compute "sum of square error" again

Step 4: Repeat steps 1-3 until termination condition is met, resetting Δw to 0

Stochastic gradient descent

Gradient descent uses batch update (update after "seeing" all samples)
↳ Difficulty #1: Slow convergence to local minimum
- ▷ incremental gradient descent, or
- ▷ stochastic gradient descent (update after "seeing" a random sample / subset of samples)

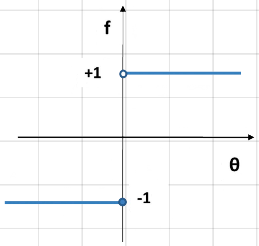
↳ Difficulty #2: No guaranteed convergence to global minimum
- ▷ Run the gradient descent algorithm repeatedly, with different seeds

Activation function /gain function / transfer function / squashing function

"To ensure that output range is restricted to $[0,1]$, differentiable activation fn is used. The computation of gradient of error must take into account the derivative of the differentiable activation fn. Weight is corrected by negative of gradient: $\sum_i (t_i - o_i) o_i (1-o_i) x_i$. E.g. if sigmoid fn is used, $o_i = \sigma(x_i w)$."

Example 1: Sign function ⇒ non-differentiable

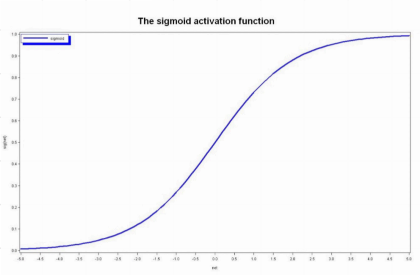
$$f(\theta) = \begin{cases} 1, & \theta > 0 \\ -1, & \theta \leq 0 \end{cases}$$

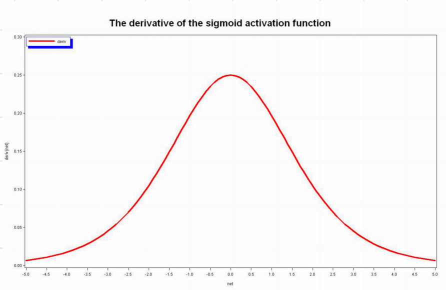


Example 2. Sigmoid function ⇒ differentiable

$$\sigma(y) = \frac{1}{1+e^{-y}} \quad \text{where } y = w^T x$$

$$\sigma'(y) = \sigma(y)(1-\sigma(y))$$

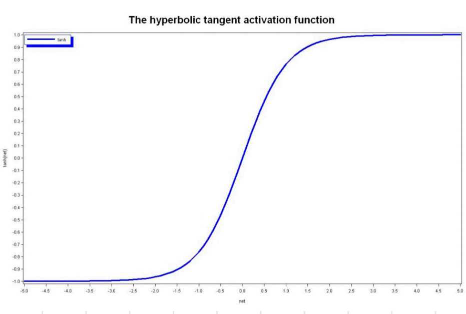


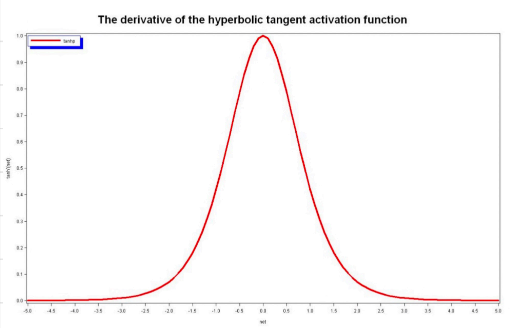


Example 3: hyperbolic tangent ⇒ differentiable

$$\tanh(y) = \frac{e^{y} - e^{-y}}{e^{y} + e^{-y}} = 2\sigma(2y) - 1 \quad \text{where } y = w^T x$$

$$\tanh'(y) = 1 - \tanh(y)^2$$





Section 1 : Training single-layer perceptron : gradient descent

Section 2 : Training multi-layer perceptron : back-propagation

Back-propagation

Note : When using multi-layer perceptrons, a differentiable activation function is required after each weighted sum calculation. Else, because linear combination of linear combinations is still a linear combination, having multiple layers does not add value to the model. (Without activation function , model is performing linear regression at each step ; with activation function , it is performing logistic regression)

Initialisation : Set $E_{max} > 0$, $\eta > 0$, W to a random value, V to a random value. Initialise $E = 0$.

Optimization : For each input data , perform Phase I & Phase II .

Termination : If $E < E_{max}$, stop . Else , $E = 0$, go to step 2 .

Notation

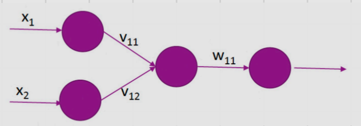
- Each training example is denoted as (x, d) where x is the vector of input values, d is the vector of target network values
- η : learning rate
- I, J, K : # input units, hidden units, output units
- W : weight matrix for connections from hidden units to output units , with K rows & J columns
- V . weight matrix for connections from input units to hidden units , with J rows & I columns
- y : vector of hidden unit activations , with J rows
- o : vector of output unit activations , with K rows

Phase I (feedforward phase)

Step 1 : Compute hidden unit activation & output unit activation

- hidden unit activation , $y_j = f(v_j^T x)$ for $j = 1, 2, \dots, J$
- output unit activation , $o_k = f(w_k^T y)$ for $k = 1, 2, \dots, K$

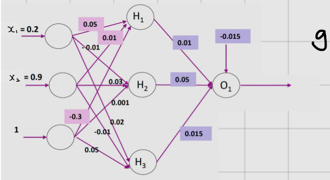
Example 1



$I = 2$, $J = 1$, $K = 1$

- Suppose activation function used at hidden & output units is sigmoid, given input data $\binom{x_1}{x_2}$,
  - ↳ hidden unit activation : $y_1 = \sigma(v^T x) = \sigma(v_{11} x_1 + v_{12} x_2)$
  - ↳ predicted output : $o_1 = \sigma(w^T y) = \sigma(w_{11} y_1)$
  - ↳ prediction error $= (d_1 - o_1)$, contribution to error sum $= \frac{1}{2}(d_1 - o_1)^2$

Example 2



- Suppose activation function used at hidden & output units is sigmoid, given input data $\binom{x_1}{x_2}$,
  - ↳ At H_1 , $w_1 x_1 + w_2 x_2 + w_3 x_3 = (0.05)(0.2) + (0.01)(0.9) - 0.3 = -0.281$
    activation value $= \frac{1}{1 + e^{0.281}} = 0.431$
  - ↳ At O_1 , $w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{bias} = (0.01)(0.430) + (0.05)(0.506) + (0.015)(0.511) - 0.015 = 0.0$
    activation value $= \frac{1}{1 + e^{-0.0233}} = 0.506$
    If cut-off $= 0.5$, the sample is classified as Class 1

Step 2 : Compute prediction error

- $E += \frac{1}{2}(d_k - o_k)^2$

Phase II (backpropagation phase)

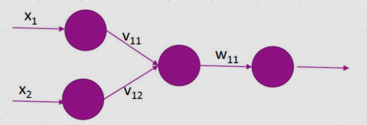
Step 1 : Compute error signal vectors δ_o and δ_y for output and hidden layer units

- $\delta_{ok} = (d_k - o_k)(1 - o_k)o_k$ for $k = 1, 2, \dots, K$
- $\delta_{y_j} = y_j(1-y_j)\sum_{k=1}^{K} \delta_{ok}\, w_{kj}$ for $j = 1, 2, \dots, J$

Step 2 : Update output layer and hidden layer weights

- $w_{kj} \mathrel{+}= \eta\, \delta_{ok}\, y_j$, where $\frac{\partial E}{\partial w_{kj}} = -\delta_{ok}\, y_j$ for $k = 1, 2, \dots, K$ and $j = 1, 2, \dots, J$
- $v_{ji} \mathrel{+}= \eta\, \delta_{y_j}\, x_i$, where $\frac{\partial E}{\partial v_{ji}} = -\delta_{y_j}\, x_i$ for $j = 1, 2, \dots, J$ and $i = 1, 2, \dots, I$

Example 1



Step 1 : $\cdot\ \delta_{o1} = (d_1 - o_1)(1 - o_1)o_1$

$$\delta_{y1} = y_1(1-y_1)(d_1 - o_1)(1-o_1)o_1 w_{11}$$
$$= y_1(1-y_1)\,\delta_{o1}\, w_{11}$$

Step 2 .
- $w_{11} \mathrel{+}= \eta\, \delta_{o1}\, y_1$ $\quad \left(\frac{\partial E}{\partial w_{11}} = -\delta_{o1}\, y_1\right)$
- $v_{11} \mathrel{+}= \eta\, \delta_{y1}\, x_1$ $\quad \left(\frac{\partial E}{\partial v_{11}} = -\delta_{y1}\, x_1\right)$
- $v_{12} \mathrel{+}= \eta\, \delta_{y1}\, x_2$ $\quad \left(\frac{\partial E}{\partial v_{12}} = -\delta_{y1}\, x_2\right)$

Derivation (w/o using given formula)

$$\frac{\partial E}{\partial w_{11}} = (-1)\left(\tfrac{1}{2}\right)(2)(d_1 - o_1)\,\frac{\partial o_1}{\partial w_{11}}$$
$$= -(d_1 - o_1)\, o_1 (1 - o_1)\,\frac{\partial (w^T y)}{\partial w_{11}}$$
$$= -(d_1 - o_1)\, o_1 (1 - o_1)\, y_1$$
$$= -\delta_{o1}\, y_1$$

$$\frac{\partial E}{\partial v_{11}} = (-1)\left(\tfrac{1}{2}\right)(2)(d_1 - o_1)\,\frac{\partial o_1}{\partial v_{11}}$$
$$= -(d_1 - o_1)\, o_1 (1 - o_1)\,\frac{\partial w^T y}{\partial v_{11}}$$
$$= -(d_1 - o_1)\, o_1 (1 - o_1)\, w_{11}\,\frac{\partial y_1}{\partial v_{11}}$$
$$= -(d_1 - o_1)\, o_1 (1 - o_1)\, w_{11}\, y_1 (1 - y_1)\,\frac{\partial (v^T x)}{\partial v_{11}}$$
$$= -(d_1 - o_1)\, o_1 (1 - o_1)\, w_{11}\, y_1 (1 - y_1)\, x_1$$
$$= -\delta_{y1}\, x_1$$

$\Rightarrow$ Intuition :

To tune a weight :

compute $\frac{\partial\, \text{error}}{\partial\, \text{output}} \cdot \frac{\partial\, \text{output}}{\partial\, \text{input}} \cdot \frac{\partial\, \text{input}}{\partial\, \text{weight}}$