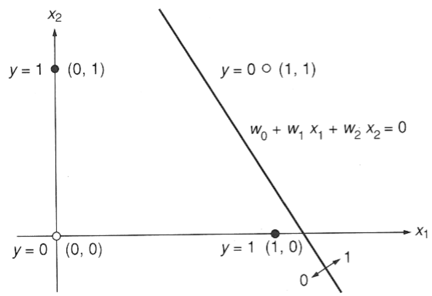
Lecture 9: Multilayer Perceptrons

Andreas Wichert
Department of Computer Science and Engineering
Técnico Lisboa

1

XOR problem and Perceptron

- By Minsky and Papert in mid 1960



2

Multi-layer Networks

- The limitations of simple perceptron do not apply to feed-forward networks with intermediate or „hidden“ nonlinear units
- A network with just one hidden unit can represent any Boolean function
- The great power of multi-layer networks was realized long ago
  - But it was only in the eighties it was shown how to make them learn

3

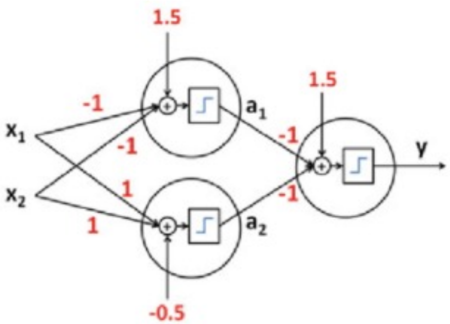
XOR-example

Knowl Inf Syst (2012) 30:135–154
DOI 10.1007/s10115-011-0392-6

REGULAR PAPER

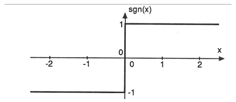
A general insight into the effect of neuron structure on classification

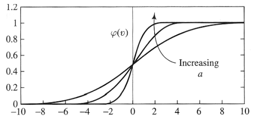
Hadi Sadoghi Yazdi · Alireza Rowhanimanesh · Hamidreza Modares



4

- Multiple layers of cascade linear units still produce only linear functions
- We search for networks capable of representing nonlinear functions
  - Units should use nonlinear activation functions
  - Examples of nonlinear activation functions





5

Gradient Descent for one Unit

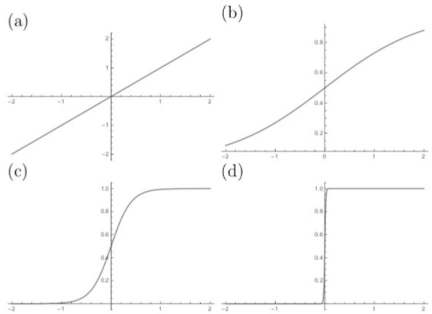


Figure 1.1: (a) Linear activation function. (b) The function $\sigma(net)$ with $\alpha = 1$. (c) The function $\sigma(net)$ with $\alpha = 5$. (d) The function $\sigma(net)$ with $\alpha = 10$ is very similar to $sgn_0(net)$, bigger α make it even more similar.

6

Linear Unit

$$o_k = \sum_{j=0}^{D} w_j \cdot x_{k,j}$$

The update rule for gradient decent is given by

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

7

Sigmoid Unit

$$\sigma(net) = \frac{1}{1 + e^{(-\alpha \cdot net)}} = \frac{e^{(\alpha \cdot net)}}{1 + e^{(\alpha \cdot net)}}$$

$$o_k = \sigma\left(\sum_{j=0}^{N} w_j \cdot x_{k,j}\right)$$

$$\frac{\partial E}{\partial w_j} = -\alpha \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot \sigma\left(net_{k,j}\right) \cdot \left(1 - \sigma\left(net_{k,j}\right)\right) \cdot x_{k,j}.$$

$$\Delta w_j = \eta \cdot \alpha \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot \sigma\left(net_{k,j}\right) \cdot \left(1 - \sigma\left(net_{k,j}\right)\right) \cdot x_{k,j}$$

8

Logistic Regression

$$p(C_1|\mathbf{x}) = \sigma(net) = \frac{1}{1+e^{(-net)}} = \frac{e^{(net)}}{1+e^{(net)}}$$

$$p(C_1|\mathbf{x}) = \sigma\left(\sum_{j=0}^{N} w_j \cdot x_j\right) = \sigma\left(\mathbf{w}^T \cdot \mathbf{x}\right)$$

Error function is defined by negative logarithm of the likelihood which leads to the update rule where the target t_k can be only one or zero (a constraint)

The update rule for gradient decent is given for target $t_k \in \{0,1\}$

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

9

Back-propagation (1980)

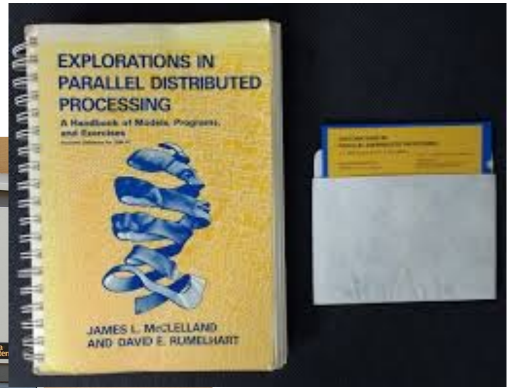
- Back-propagation is a learning algorithm for multi-layer neural networks
- It was invented independently several times
  - Bryson an Ho [1969]
  - Werbos [1974]
  - Parker [1985]
  - **Rumelhart et al. [1986]**

Parallel Distributed Processing - Vol. 1
Foundations
David E. Rumelhart, James L. McClelland and the PDP Research Group

What makes people smarter than computers? These volumes by a pioneering neurocomputing.....

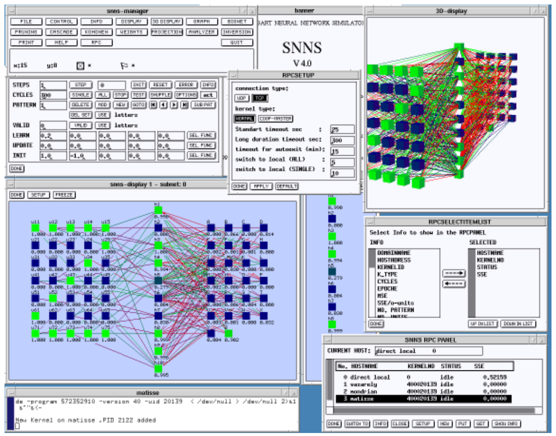
10

The good old days…



11

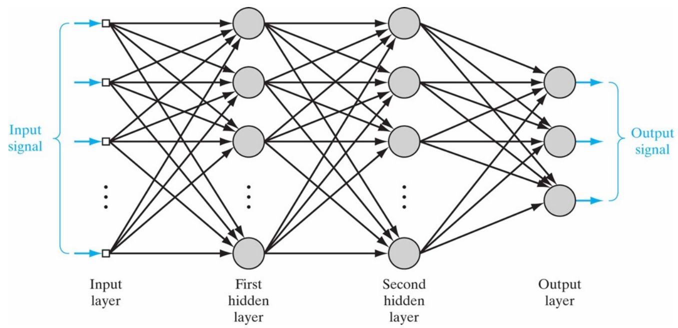
Everyone was doing Back-propagation….



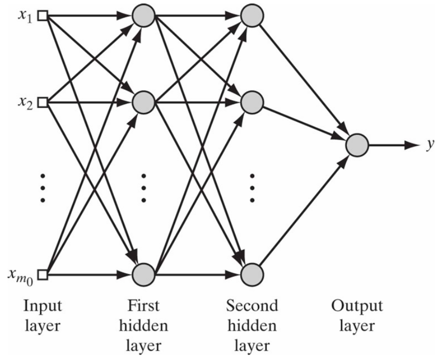
12

- Feed-forward networks with hidden nonlinear units are universal approximators; they can approximate every bounded continuous function with an arbitrarily small error
- Each Boolean function can be represented by a network with a single hidden layer
  - However, the representation may require an exponential number of hidden units.
- The hidden units should be nonlinear because multiple layers of linear units can only produce linear functions.

13



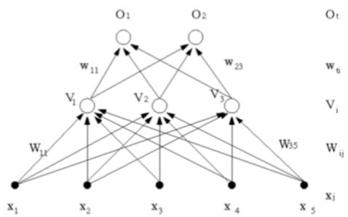
14



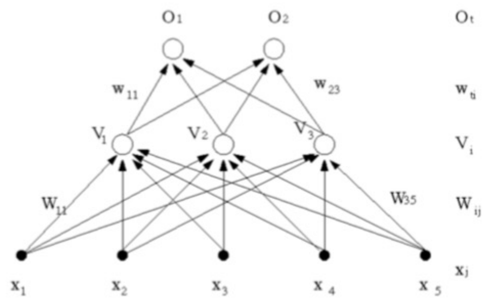
15

Back-propagation

- The algorithm gives a prescription for changing the weights w_{ij} in any feed-forward network to learn a training set of input output pairs $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}$
- We consider a simple two-layer network



16



- The input pattern is represented by the five-dimensional vector ***x***
- nonlinear hidden units compute the output V_1, V_2, V_3
- Two output units compute the output o_1 and o_2.
  - The units V_1, V_2, V_3 are referred to as hidden units because we cannot see their outputs and cannot directly perform error correction

17

- The output layer of a feed-forward network can be trained by the perceptron rule (stochastic gradient descent) since it is a Perceptron

$$\Delta w_{ti} = \eta \cdot (y_{k,t} - o_{k,t}) \cdot V_{k,i}.$$

For continuous activation function $\phi()$

$$o_{k,t} = \phi\left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i}\right).$$

18

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{t=1}^{2} \sum_{k=1}^{N} (y_{kt} - o_{kt})^2 = \frac{1}{2} \cdot \sum_{t=1}^{2} \sum_{k=1}^{N} \left(y_{kt} - \phi \left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i} \right) \right)^2$$

we get

$$\frac{\partial E}{\partial w_{ti}} = - \sum_{k=1}^{N} (y_{kt} - o_{kt}) \cdot \phi' \left(\sum_{i=0}^{n} w_{ti} \cdot V_{k,i} \right) \cdot V_{k,i}.$$

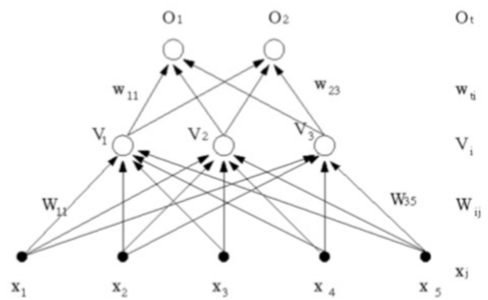
For the nonlinear continuous function $\sigma()$

$$\frac{\partial E}{\partial w_{ti}} = -\alpha \cdot \sum_{k=1}^{N} (y_{kt} - o_{kt}) \cdot \sigma(net_{k,t}) \cdot (1 - \sigma(net_{k,t})) \cdot V_{k,i}$$

and

$$\Delta w_{ti} = \eta \cdot \alpha \cdot \sum_{k=1}^{N} (y_{kt} - o_{kt}) \cdot \sigma(net_{k,t}) \cdot (1 - \sigma(net_{k,t})) \cdot V_{k,i}.$$

19



We can determine the Δw_{ti} for the output units, but how can we determine ΔW_{ij} for the hidden units? If the hidden units use a continuous non linear activation function $\phi()$

$$V_{k,i} = \phi \left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j} \right).$$

20

$$V_{k,i} = \phi\left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j}\right).$$

we can define the training error for a training data set D_t of N elements with

$$E(\mathbf{w}, \mathbf{W}) =: E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} - o_{kt})^2$$

$$E(\mathbf{w}, \mathbf{W}) = \frac{1}{2} \cdot \sum_{k=1}^{N} \sum_{t=1}^{2} \left(y_{kt} - \phi\left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i}\right)\right)^2$$

$$E(\mathbf{w}, \mathbf{W}) = \frac{1}{2} \cdot \sum_{k=1}^{N} \sum_{t=1}^{2} \left(y_{kt} - \phi\left(\sum_{i=0}^{3} w_{ti} \cdot \phi\left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j}\right)\right)\right)^2.$$

21

We already know

$$\frac{\partial E}{\partial w_{ti}} = -\sum_{k=1}^{N} (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t}) \cdot V_{k,i}.$$

For $\frac{\partial E}{\partial W_{ij}}$ we can use the chain rule and we obtain

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k=1}^{N} \frac{\partial E}{\partial V_{ki}} \cdot \frac{\partial V_{ki}}{\partial W_{ij}}.$$

22

$$E(\mathbf{w}, \mathbf{W}) = \frac{1}{2} \cdot \sum_{k=1}^{N} \sum_{t=1}^{2} \left(y_{kt} - \phi \left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i} \right) \right)^2$$

$$E(\mathbf{w}, \mathbf{W}) = \frac{1}{2} \cdot \sum_{k=1}^{N} \sum_{t=1}^{2} \left(y_{kt} - \phi \left(\sum_{i=0}^{3} w_{ti} \cdot \phi \left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j} \right) \right) \right)^2 .$$

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k=1}^{N} \frac{\partial E}{\partial V_{ki}} \cdot \frac{\partial V_{ki}}{\partial W_{ij}}.$$

with

$$\frac{\partial E}{\partial V_{ki}} = -\sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t}) \cdot w_{t,i}.$$

$$V_{k,i} = \phi \left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j} \right) \longrightarrow \frac{\partial V_{ki}}{\partial W_{ij}} = \phi'(net_{k,i}) \cdot x_{k,j}$$

23

$$\frac{\partial E}{\partial V_{ki}} = -\sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t}) \cdot w_{t,i}.$$

$$\frac{\partial V_{ki}}{\partial W_{ij}} = \phi'(net_{k,i}) \cdot x_{k,j}$$

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k=1}^{N} \frac{\partial E}{\partial V_{ki}} \cdot \frac{\partial V_{ki}}{\partial W_{ij}}.$$

$$\frac{\partial E}{\partial W_{ij}} = -\sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t}) \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}.$$

24

The algorithm is called back propagation because we can reuse the computation that was used to determine Δw_{ti},

$$\Delta w_{ti} = \eta \cdot \sum_{k=1}^{N} (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t}) \cdot V_{k,i}.$$

and with

$$\delta_{kt} = (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t})$$

we can write

$$\Delta w_{ti} = \eta \cdot \sum_{k=1}^{N} \delta_{kt} \cdot V_{k,i}.$$

25

$$\Delta W_{ij} = \eta \sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t}) \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}$$

we can simplify (reuse the computation) to

$$\delta_{kt} = (y_{kt} - o_{kt}) \cdot \phi'(net_{k,t})$$

$$\Delta W_{ij} = \eta \sum_{k=1}^{N} \sum_{t=1}^{2} \delta_{kt} \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}.$$

With

$$\delta_{ki} = \phi'(net_{k,i}) \cdot \sum_{t=1}^{2} \delta_{kt} \cdot w_{t,i}$$

we can simply to

$$\Delta W_{ij} = \eta \sum_{k=1}^{N} \delta_{ki} \cdot x_{k,j}.$$

26

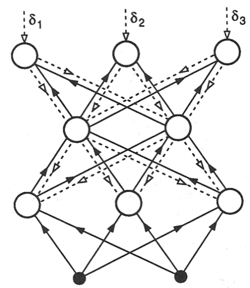
- In general, with an arbitrary number of layers, the back-propagation update rule has always the form

$$\Delta w_{ij} = \eta \sum_{d=1}^{m} \delta_{output} \cdot V_{input}$$

  - Where output and input refers to the connection concerned
  - V stands for the appropriate input (hidden unit or real input, $\boldsymbol{x}_d$)
  - δ depends on the layer concerned

27

- This approach can be extended to any numbers of layers
- The coefficient are usual forward, but the errors represented by δ are propagated backward



28

Networks with Hidden Linear Layers

Consider simple linear unit with a linear activation function

$$o_{k,t} = \sum_{i=0}^{3} w_{ti} \cdot V_{k,i} = \mathbf{w}_t^T \cdot \mathbf{V}_k$$

$$V_{k,i} = \sum_{j=0}^{5} W_{ij} \cdot x_{k,j} = \mathbf{W}_j^T \cdot \mathbf{x}_k$$

Now W is a matrix

$$\mathbf{V}_k = W \cdot \mathbf{x}_k$$

So we can write

$$o_{k,t} = \mathbf{w}_t^T \cdot W \cdot \mathbf{x}_k$$

with

$$(\mathbf{w}_t^*)^T = \mathbf{w}_t^T \cdot W$$

and we get the same discrimination power (linear separable) as a simple Perceptron

$$o_{k,t} = (\mathbf{w}_t^*)^T \cdot \mathbf{x}_k$$

29

However with nonlinear activation function, we cannot do the matrix multiplication

$$\mathbf{V}_k = \phi\,(W \cdot \mathbf{x}_k)$$

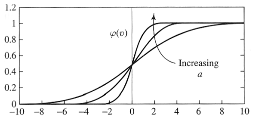
So we can write

$$o_{k,t} = \mathbf{w}_t^T \cdot \phi\,(W \cdot \mathbf{x}_k)$$

but we cannot simplify

30

- We have to use a nonlinear differentiable activation function in **hidden units**
  - Examples:



$$f(x) = \sigma(x) = \frac{1}{1 + e^{(-\alpha \cdot x)}}$$

$$f'(x) = \sigma'(x) = \alpha \cdot \sigma(x) \cdot (1 - \sigma(x))$$

$$f(x) = \tanh(\alpha \cdot x)$$

$$f'(x) = \alpha \cdot (1 - f(x)^2)$$

31

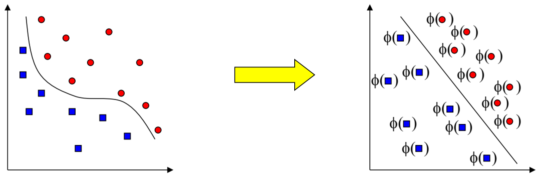
Two kind of Units

- Output Units
  - Require Bias
  - Preform **Linear Separable Problems**, means the input to them had to be somehow linearised
  - Does not require non linear activation function,
  - We **should** use sigmoid function or softmax to represent probabilities and to get **better** decision boundary.
- Hidden Units
  - Nonlinear activation function
  - Feature Extraction
    Does it require Bias? It is commonly used
  - Universal Approximation Theorem uses hidden units with bias.

32

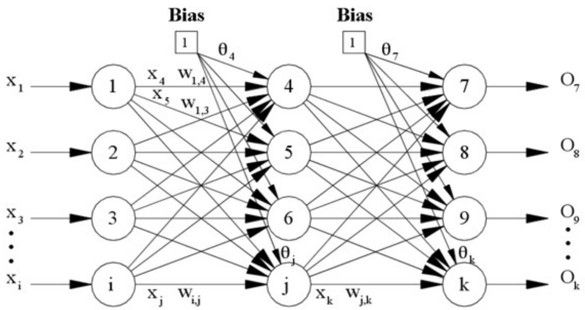
Output Units are linear (Perceptron)

- The hidden layer applies a nonlinear transformation from the input space to the hidden space
- In the hidden space a linear discrimination can be performed



33

Bias?



34

More on Back-Propagation

- Gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)

35

- Gradient descent can be very slow if η is to small, and can oscillate widely if η is to large
- Often include weight *momentum* α

$$\Delta w_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \alpha \cdot \Delta w_{pq}(t)$$

- *Momentum* parameter α is chosen between *0* and *1, 0.9* is a good value

36

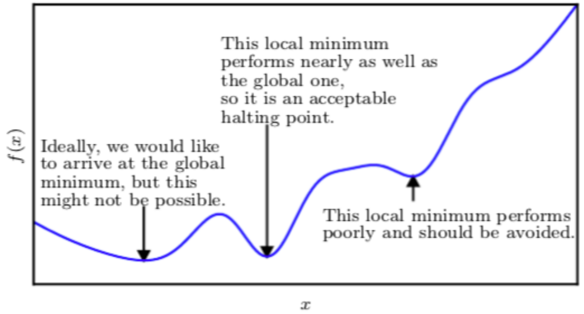
- Minimizes error over training examples
  - Will it generalize well
- Training can take thousands of iterations, it is slow!
- Using network after training is very fast

37

Convergence of Back-propagation

- Gradient descent to some local minimum
  - Perhaps not global minimum...
  - Add momentum
  - Stochastic gradient descent
  - Train multiple nets with different initial weights
- Nature of convergence
  - Initialize weights near zero
  - Therefore, initial networks near-linear
  - Increasingly non-linear functions possible as training progresses

38

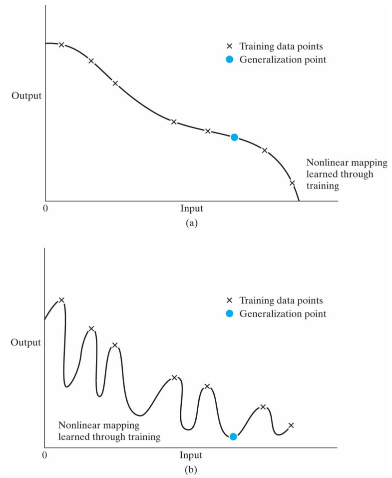


39

Expressive Capabilities of ANNs

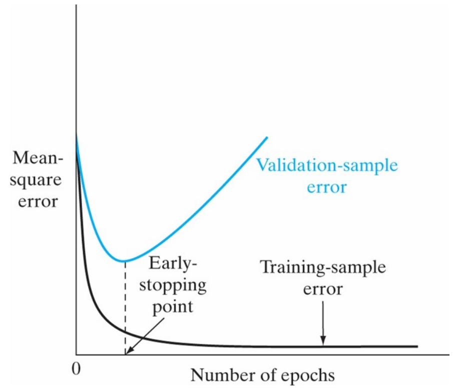
- Boolean functions:
  - Every boolean function can be represented by network with single hidden layer
  - but might require exponential (in number of inputs) hidden units
- Continuous functions:
  - Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
    - See: https://en.wikipedia.org/wiki/Universal_approximation_theorem
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

40

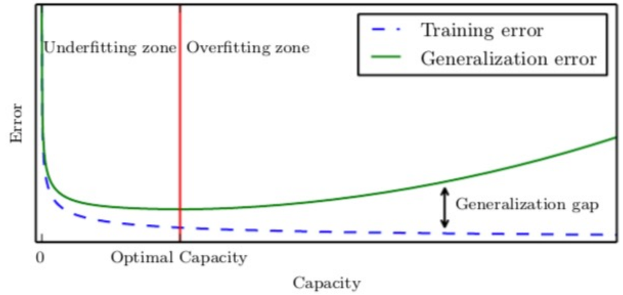


41

Early-Stopping Rule

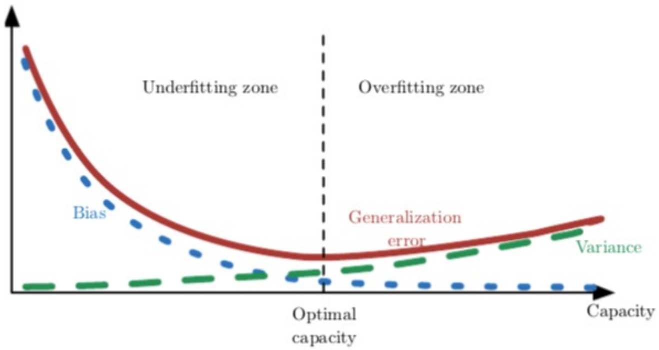


42



Underfitting zone
Overfitting zone
Training error
Generalization error
Error
Generalization gap
0
Optimal Capacity
Capacity

43



Underfitting zone
Overfitting zone
Bias
Generalization error
Variance
Optimal capacity
Capacity

44

Example
(different notation)

- Consider a network with M layers $m=1,2,..,M$
- V^m_i from the output of the ith unit of the mth layer
- V^0_i is a synonym for x_i of the ith input
- Subscript m layers m's layers, not patterns
- W^m_{ij} mean connection from V_j^{m-1} to V_i^m

45

$$\Delta W_{ij} = \eta \sum_{d=1}^{m} \delta_i^d V_j^d$$

$$\Delta w_{jk} = \eta \sum_{d=1}^{m} \delta_j^d \cdot x_k^d$$

- We have same form with a different definition of δ
- d is the pattern identificator

46

- By the equation $\delta_j^d = f'(net_j^d)\sum_{i=1}^{2} W_{ij}\delta_i^d$
- allows us to determine for a given hidden unit V_j in terms of the δ's of the unit o_i
- The coefficient are usual forward, but the errors δ are propagated backward
  - back-propagation

47

Stochastic Back-Propagation Algorithm

(mostly used)

1. Initialize the weights to small random values
2. Choose a pattern x^d_k and apply is to the input layer $V^0_k = x^d_k$ for all k
3. Propagate the signal through the network

   $$V_i^m = f(net_i^m) = f\left(\sum_j w_{ij}^m V_j^{m-1}\right)$$

4. Compute the deltas for the output layer

   $$\delta_i^M = f'(net_i^M)(t_i^d - V_i^M)$$

5. Compute the deltas for the preceding layer for $m=M,M-1,..2$

   $$\delta_i^{m-1} = f'(net_i^{m-1})\sum_j w_{ji}^m \delta_j^m$$

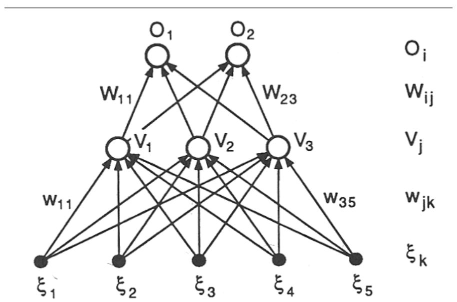
6. Update all connections

   $$\Delta w_{ij}^m = \eta\delta_i^m V_j^{m-1} \qquad w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$

7. Goto 2 and repeat for the next pattern

48

Example



w_1 $=\{w_{11}=0.1, w_{12}=0.1, w_{13}=0.1, w_{14}=0.1, w_{15}=0.1\}$

w_2 $=\{w_{21}=0.1, w_{22}=0.1, w_{23}=0.1, w_{24}=0.1, w_{25}=0.1\}$

w_3 $=\{w_{31}=0.1, w_{32}=0.1, w_{33}=0.1, w_{34}=0.1, w_{35}=0.1\}$

W_1 $=\{W_{11}=0.1, W_{12}=0.1, W_{13}=0.1\}$

W_2 $=\{W_{21}=0.1, W_{22}=0.1, W_{23}=0.1\}$

X_1 $=\{1,1,0,0,0\}$; ***t_1***$=\{1,0\}$

X_2 $=\{0,0,0,1,1\}$; ***t_1***$=\{0,1\}$

$$f(x) = \sigma(x) = \frac{1}{1+e^{(-x)}} \qquad f'(x) = \sigma'(x) = \sigma(x)\cdot(1-\sigma(x))$$

49

$$net_1^1 = \sum_{k=1}^{5} w_{1k} x_k^1 \qquad V_1^1 = f(net_1^1) = \frac{1}{1+e^{-net_1^1}}$$

*net^1_1=1*0.1+1*0.1+0*0.1+0*0.1+0*0.1*

V^1_1=f(net^1_1)=1/(1+exp(-0.2))=0.54983

$$net_2^1 = \sum_{k=1}^{5} w_{2k} x_k^1 \qquad V_2^1 = f(net_1^1) = \frac{1}{1+e^{-net_2^1}}$$

V^1_2=f(net^1_2)=1/(1+exp(-0.2))=0.54983

$$net_3^1 = \sum_{k=1}^{5} w_{3k} x_k^1 \qquad V_3^1 = f(net_3^1) = \frac{1}{1+e^{-net_3^1}}$$

V^1_3=f(net^1_3)=1/(1+exp(-0.2))=0.54983

50

$$net_1^1 = \sum_{j=1}^{3} W_{1j} V_j^1 \qquad o_1^1 = f(net_1^1) = \frac{1}{1+e^{-net_1^1}}$$

*net¹₁=0.54983*0.1+ 0.54983*0.1+ 0.54983*0.1= 0.16495*

o¹₁= f(net11)=1/(1+exp(- 0.16495))= 0.54114

$$net_2^1 = \sum_{j=1}^{3} W_{2j} V_j^1 \qquad o_2^1 = f(net_2^1) = \frac{1}{1+e^{-net_2^1}}$$

*net¹₂=0.54983*0.1+ 0.54983*0.1+ 0.54983*0.1= 0.16495*

o¹₂= f(net11)=1/(1+exp(- 0.16495))= 0.54114

51

For hidden-to-output

$$\Delta W_{ij} = \eta \sum_{d=1}^{m} (t_i^d - o_i^d) f'(net_i^d) \cdot V_j^d$$

- We will use **stochastic gradient** descent with *η=1*

$$\Delta W_{ij} = (t_i - o_i) f'(net_i) V_j$$

$$f'(x) = \sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

$$\Delta W_{ij} = (t_i - o_i)\sigma(net_i)(1 - \sigma(net_i))V_j$$

$$\delta_i = (t_i - o_i)\sigma(net_i)(1 - \sigma(net_i))$$

$$\Delta W_{ij} = \delta_i V_j$$

52

$$\delta_1 = (t_1 - o_1)\sigma(net_1)(1 - \sigma(net_1))$$
$$\Delta W_{1j} = \delta_1 V_j$$

- *δ_1=(1- 0.54114)*(1/(1+exp(- 0.16495)))*(1-(1/(1+exp(- 0.16495))))= 0.11394*

$$\delta_2 = (t_2 - o_2)\sigma(net_2)(1 - \sigma(net_2))$$
$$\Delta W_{2j} = \delta_2 V_j$$

- *δ_2=(0- 0.54114)*(1/(1+exp(- 0.16495)))*(1-(1/(1+exp(- 0.16495))))= -0.13437*

53

Input-to hidden connection

$$\Delta w_{jk} = \sum_{i=1}^{2} \delta_i \cdot W_{ij} f'(net_j) \cdot x_k$$

$$\Delta w_{jk} = \sum_{i=1}^{2} \delta_i \cdot W_{ij} \sigma(net_j)(1 - \sigma(net_j)) \cdot x_k$$

$$\delta_j = \sigma(net_j)(1 - \sigma(net_j)) \sum_{i=1}^{2} W_{ij} \delta_i$$

$$\Delta w_{jk} = \delta_j \cdot x_k$$

54

$$\delta_1 = \sigma(net_1)(1-\sigma(net_1))\sum_{i=1}^{2} W_{i1}\delta_i$$

δ_1= 1/(1+exp(- 0.2))*(1- 1/(1+exp(- 0.2)))*(0.1* 0.11394+0.1*(-0.13437))

δ_1= -5.0568e-04

$$\delta_2 = \sigma(net_2)(1-\sigma(net_2))\sum_{i=1}^{2} W_{i2}\delta_i$$

δ_2= -5.0568e-04

$$\delta_3 = \sigma(net_3)(1-\sigma(net_3))\sum_{i=1}^{2} W_{i3}\delta_i$$

δ_3= -5.0568e-04

55

First Adaptation for $\boldsymbol{x_1}$

(one epoch, adaptation over all training patterns, in our case $\boldsymbol{x_1}$ $\boldsymbol{x_2}$)

$$\Delta w_{jk} = \delta_j \cdot x_k$$

$$\Delta W_{ij} = \delta_i V_j$$

δ_1= -5.0568e-04	δ_1= 0.11394
δ_2= -5.0568e-04	δ_2= -0.13437
δ_3= -5.0568e-04	
x_1 =1	v_1 =0.54983
x_2 =1	v_2 =0.54983
x_3 =0	v_3=0.54983
x_4 =0	
x_5 =0	

56

Learning consists of minimizing the error (loss) function [Bishop, 2006],

$$E(\mathbf{w}) = -\sum_{k=1}^{N} y_k \log o_k$$

in which $y_{kt} \in \{0,1\}$ and o_k corresponds to probabilities ($\sum_t y_{kt} = 1$). The error surface is more steeply as the error surface defined by the squared error

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} - o_{kt})^2$$

and the gradient converges faster. The cross entropy error function can be alternatively written as loss (cost) function with $\theta = \mathbf{w}$

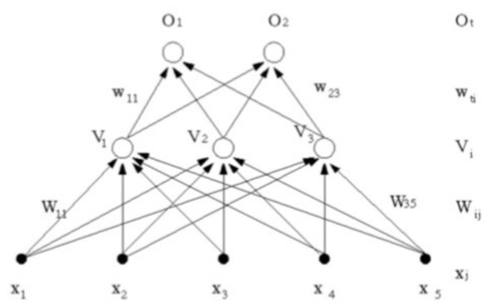
$$L(\mathbf{x}, \mathbf{y}, \theta) = -\sum_{k=1}^{N} (y_k \log p(c_k|\mathbf{x}))$$

or as the loss function

$$J(\theta) = -\sum_{k=1}^{N} (y_k \log p(c_k|\mathbf{x})) = -\mathbb{E}_{x,y\sim p_{data}} \log p(c_k|\mathbf{x})$$

in which θ indicates the adaptive parameters of the model and $\mathbb{E}$ indicates the expectation. This notation is usually common in statistics.

57



- The input pattern is represented by the five-dimensional vector ***x***
- nonlinear hidden units compute the output V_1, V_2, V_3
- Two output units compute the output o_1 and o_2.
  - The units V_1, V_2, V_3 are referred to as hidden units because we cannot see their outputs and cannot directly perform error correction

58

For simplicity we define ϕ as a sigmoid function.

For output layer it is the softmax function with

$$\phi(net) = \frac{\exp(net_k)}{\sum_{j=1}^{K} \exp(net_j)}$$

For the hidden units it is

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

We can use different activation function, using the sigmoid function we can reuse the results which we developed when we introduced the logistic regression

We assume the target values $y_{kt} \in \{0, 1\}$

59

We assume the target values $y_{kt} \in \{0, 1\}$

Output unit

$$o_{k,t} = \phi\left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i}\right).$$

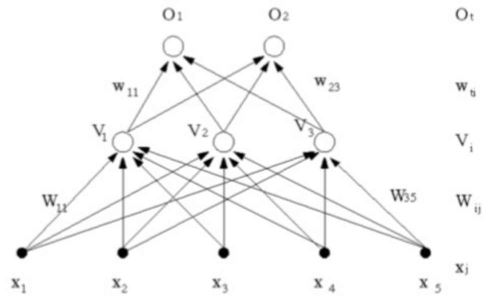
and

$$E(\mathbf{w}) = -\sum_{t=1}^{2}\sum_{k=1}^{N} y_{kt} \log o_{kt} = \sum_{t=1}^{2}\sum_{k=1}^{N} y_{kt} \log \phi\left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i}\right)$$

we get (logistic regression)

$$\frac{\partial E}{\partial w_{ti}} = -\sum_{k=1}^{N} (y_{kt} - o_{kt}) \cdot V_{k,i}.$$

60



We can determine the Δw_{ti} for the output units, but how can we determine ΔW_{ij} for the hidden units? If the hidden units use a continuous non linear activation function $\phi()$

$$V_{k,i} = \phi\left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j}\right).$$

61

$$V_{k,i} = \phi\left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j}\right).$$

we can define the training error for a training data set D_t of N elements with

$$E(\mathbf{w}, \mathbf{W}) =: E(\mathbf{w}) = -\sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} \cdot \log o_{kt})$$

$$E(\mathbf{w}, \mathbf{W}) = -\sum_{k=1}^{N} \sum_{t=1}^{2} \left(y_{kt} \cdot \log \phi\left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i}\right)\right)$$

$$E(\mathbf{w}, \mathbf{W}) = -\sum_{k=1}^{N} \sum_{t=1}^{2} \left(y_{kt} \cdot \log \phi\left(\sum_{i=0}^{3} w_{ti} \cdot \phi\left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j}\right)\right)\right)$$

62

We already know

$$\frac{\partial E}{\partial w_{ti}} = -\sum_{k=1}^{N}(y_{kt} - o_{kt}) \cdot V_{k,i}.$$

For $\frac{\partial E}{\partial W_{ij}}$ we can use the chain rule and we obtain

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k=1}^{N} \frac{\partial E}{\partial V_{ki}} \cdot \frac{\partial V_{ki}}{\partial W_{ij}}.$$

63

$$E(\mathbf{w}, \mathbf{W}) = -\sum_{k=1}^{N}\sum_{t=1}^{2}\left(y_{kt} \cdot \log\phi\left(\sum_{i=0}^{3} w_{ti} \cdot V_{k,i}\right)\right)$$

$$E(\mathbf{w}, \mathbf{W}) = -\sum_{k=1}^{N}\sum_{t=1}^{2}\left(y_{kt} \cdot \log\phi\left(\sum_{i=0}^{3} w_{ti} \cdot \phi\left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j}\right)\right)\right)$$

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k=1}^{N} \frac{\partial E}{\partial V_{ki}} \cdot \frac{\partial V_{ki}}{\partial W_{ij}}.$$

$$\frac{\partial E}{\partial V_{ki}} = -\sum_{k=1}^{N}\sum_{t=1}^{2}(y_{kt} - o_{kt}) \cdot w_{t,i}.$$

$$V_{k,i} = \phi\left(\sum_{j=0}^{5} W_{ij} \cdot x_{k,j}\right) \longrightarrow \frac{\partial V_{ki}}{\partial W_{ij}} = \phi'(net_{k,i}) \cdot x_{k,j}$$

64

$$\frac{\partial E}{\partial W_{ij}} = \sum_{k=1}^{N} \frac{\partial E}{\partial V_{ki}} \cdot \frac{\partial V_{ki}}{\partial W_{ij}}.$$

with

$$\frac{\partial E}{\partial V_{ki}} = -\sum_{k=1}^{N}\sum_{t=1}^{2}(y_{kt} - o_{kt}) \cdot w_{t,i}.$$

and

$$\frac{\partial V_{ki}}{\partial W_{ij}} = \phi'(net_{k,i}) \cdot x_{k,j}$$

it follows

$$\frac{\partial E}{\partial W_{ij}} = -\sum_{k=1}^{N}\sum_{t=1}^{2}(y_{kt} - o_{kt}) \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}.$$

65

$$\frac{\partial E}{\partial W_{ij}} = -\sum_{k=1}^{N}\sum_{t=1}^{2}(y_{kt} - o_{kt}) \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}.$$

For the quadratic error it was

$$\frac{\partial E}{\partial W_{ij}} = -\sum_{k=1}^{N}\sum_{t=1}^{2}(y_{kt} - o_{kt}) \cdot \phi'(net_{k,t}) \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}.$$

You notice he difference that makes the convergence faster?

66

The algorithm is called back propagation because we can reuse the computation that was used to determine Δw_{ti},

$$\Delta w_{ti} = \eta \cdot \sum_{k=1}^{N} (y_{kt} - o_{kt}) \cdot V_{k,i}.$$

and with

$$\delta_{kt} = (y_{kt} - o_{kt})$$

we can write

$$\Delta w_{ti} = \eta \cdot \sum_{k=1}^{N} \delta_{kt} \cdot V_{k,i}.$$

67

$$\Delta W_{ij} = \eta \sum_{k=1}^{N} \sum_{t=1}^{2} (y_{kt} - o_{kt}) \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}$$

we can simplify (reuse the computation) to

$$\Delta W_{ij} = \eta \sum_{k=1}^{N} \sum_{t=1}^{2} \delta_{kt} \cdot w_{t,i} \cdot \phi'(net_{k,i}) \cdot x_{k,j}.$$

With

$$\delta_{ki} = \phi'(net_{k,i}) \cdot \sum_{t=1}^{2} \delta_{kt} \cdot w_{t,i}$$

we can simply to

$$\Delta W_{ij} = \eta \sum_{k=1}^{N} \delta_{ki} \cdot x_{k,j}.$$

68

Cross-entropy vs. Quadratic loss

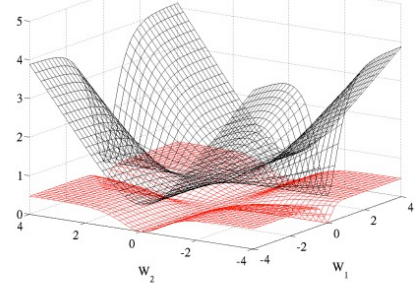


Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Figure from Glorot & Bentio (2010)

69

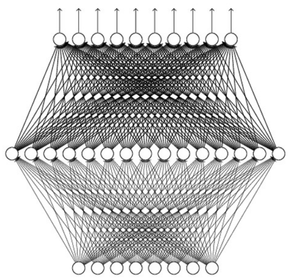
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

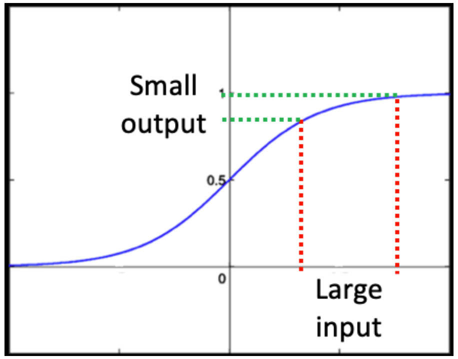
(given **enough** hidden neurons)



Why "Deep" neural network not "Fat" neural network?

70

Vanishing Gradient Problem, sigmoid

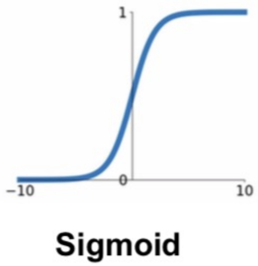


71

Sigmoid

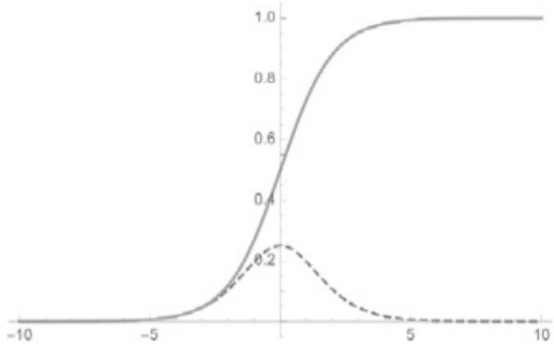
$$f(x) = \sigma(x) = \frac{1}{1 + e^{(-\alpha \cdot x)}}$$

$$f^{'}(x) = \sigma^{'}(x) = \alpha \cdot \sigma(x) \cdot (1 - \sigma(x))$$



- Squashes numbers to range [0,1]
- Historically popular
- Have nice interpretation as a saturating “firing rate” of a neuron
- 3 problems:
  - Saturated neurons “kill” the gradients
  - Sigmoid outputs are not zero-centered
  - exp() is a bit compute expensive

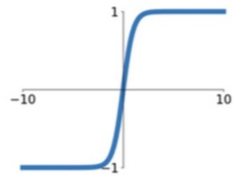
72



- The sigmoid function and the derivative indicated by doted line.

73

$$f(x) = \tanh(\alpha \cdot x)$$
$$f^{'}(x) = \alpha \cdot (1 - f(x)^2)$$



tanh(x)

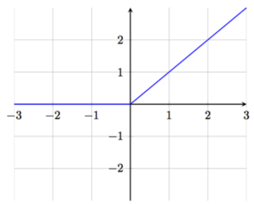
- Squashes numbers to range [-1,1]
- - zero centered (nice)
- - kills gradients when saturated ☹

74

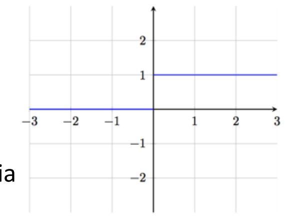
Rectified Linear Unit (ReLU)

- *f(x) = max(0,x)*
  - Function defined as the positive part of its argument
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- More biologically plausible

- But: Not zero-centered output ☹
- Non-differentiable at zero; however it is differentia a value of 0 or 1

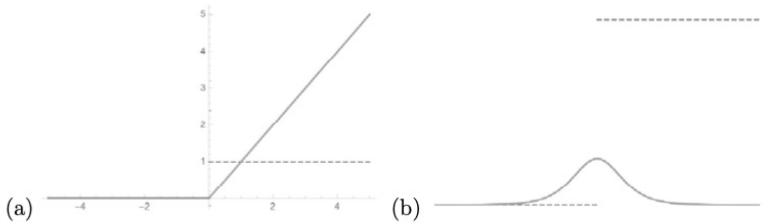


ReLu function



Derivative of ReLu function

75



- (a) Rectifier activation function (ReLU), the derivative is in- dicated by the doted line. (b) Comparing the the derivative of the sigmoid activation function and the rectifier activation function indicated by a doted line.

76

l_2 Regularization

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (y_k - o_k)^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2, \;\; or \;\; \tilde{E}(\mathbf{w}) = -\sum_{k=1}^{N} y_k \log o_k + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

$$\frac{\partial \tilde{E}}{\partial w_j} = \frac{\partial E}{\partial w_j} + \lambda \cdot w_j.$$

$$\Delta w_j = -\eta \left(\frac{\partial E}{\partial w_j} + \lambda \cdot w_j \right) = -\eta \left(\frac{\partial E}{\partial w_j} \right) - \eta\lambda \cdot w_j = -\eta \left(\frac{\partial E}{\partial w_j} \right) - \alpha \cdot w_j$$

77

l_1 Regularization

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (y_k - o_k)^2 + \lambda \cdot \|\mathbf{w}\|_1, \;\; or \;\; \tilde{E}(\mathbf{w}) = -\sum_{k=1}^{N} y_k \log o_k + \lambda \cdot \|\mathbf{w}\|_1$$

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$$

$$\frac{\partial \tilde{E}}{\partial w_j} = \frac{\partial E}{\partial w_j} + \lambda \cdot sign(w_j).$$

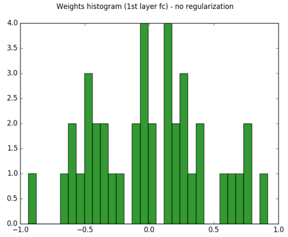
$$\Delta w_j = -\eta \left(\frac{\partial E}{\partial w_j} + \lambda \cdot sign(w_j) \right) = -\eta \left(\frac{\partial E}{\partial w_j} \right) - \eta\lambda \cdot sign(w_j)$$

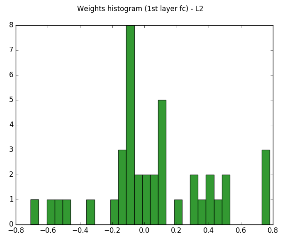
$$\Delta w_j = -\eta \left(\frac{\partial E}{\partial w_j} \right) - \alpha \cdot sign(w_j)$$

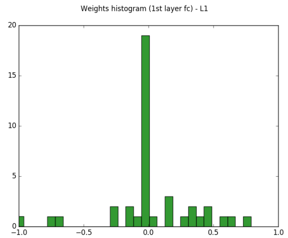
78

Overfitting

- *Example*: comparing the weights of a neural network with no regularization and l_2 and l_1 penalties

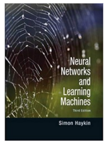




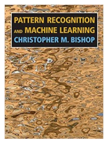


79

Literature



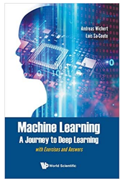
- Simon O. Haykin, Neural Networks and Learning Machine, (3rd Edition), Pearson 2008
  - Chapter 4



- Christopher M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer 2006
  - Chapter 5

80

Literature



- Machine Learning - A Journey to Deep Learning, A. Wichert, Luis Sa-Couto, World Scientific, 2021
  - Chapter 6

81