WARSAW UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS
AND INFORMATION SCIENCE

Neural Networks

Lecture 15

Solving of Optimization Problems

Optimization problems

The ability for parallel computation yields to the possibility to work with big amount of data. It allows neural networks to solve complex and time consuming optimization problems.

Optimization problems

The basic problem is to replace the task to the problem of minimization of an energy function describing the recurrent net treated as the *minimization network.*

Optimization problems

The following problems have to be solved

1. To define the problem by use of a neural network. Its final (stable) stage should determine the optimization problem
2. The energy function minimum value has to be equivalent of the solution of optimization problem

Optimization problems

3. The net structure, connection weights, threshold and out signals.
4. The elements dynamic have to assure the minimization of energy function
5. The definition of the initial values of elements

Optimization problems

- For typical combinatorial optimization problems an energy function has a form

$$E = \sum_i A_i(V_i) + B * F$$

where V_i - *is* the measure of an *i-th* constraint

F - is the objective function

A_i - and *B* are the coefficients

The TSP Optimization Problem

The „**Travelling Salesman Problem**” (TSP) is a classic of difficult optimization.

Goal:

The set of N cities *A, B, ...* have (pairwise) distance separation d_{AB}, d_{AC}, ..., d_{BC}, ...

The problem is to find a closed tour which visits each city once, returns to the starting city, and has a short (or minimum) total part length.

The TSP Optimization Problem

- There are $\frac{N!}{2N} = \frac{(N-1)!}{2}$ distinct paths for closed TSP routes and the problem is NP-hard (complete).

  To describe the N neurons in the TSP network to compute a solution to the problem, the network must be described by an energy function in which the lowest energy state (the network stable state) corresponds to the best path.

The TSP Optimization Problem

The Energy Function must be determined
in such a way that its minima correspond
to solutions of the problem considered.
The Hopfield energy function may contain several
energy terms, which may be roughly classified
into constraint terms and the objective function.

To solve by Hopfield net we need to decide the architecture:

- How many neurons?
- What are the weights?

The TSP Optimization Problem

For ***N*** cities in computer simulation the network was represented by a matrix ***N*** x ***N***.
The system is characterize using a table of cities (rows) and steps (columns).
An entry $\boldsymbol{v_{Xi}}$ is equal to **1** if city ***X*** is visited at step *i*, and **0** otherwise.

At the end of a simulation test which converged to a solution each element representing output potential of neuron $\boldsymbol{v_{Xi}}$, was equal to either zero or one.

The TSP Optimization Problem

The Hopfield network is fully interconnected, that is, all neurons are connected to all other neurons (there are no layers). The weights are symmetric.

There are three main types of weights:

a) *Each neuron has its own positive bias weight.*
b) *Each neuron has a negative (inhibitory) weight to each of the other neurons in its row and its column (α).*
c) *Each neuron has a negative weight to other possible cities just before it and just after it on the tour (d_{XY}). This weight is equal to Euclidean distance. Since the tour makes a loop, the final column is connected to the first column by distance weights.*

The TSP Optimization Problem

The energy function

$$E = \frac{A}{2}\sum_{X}\sum_{Y}\sum_{i,i\neq j} v_{Xi}v_{Xj} + \frac{B}{2}\sum_{\mathrm{i}}\sum_{X}\sum_{Y,Y\neq X} v_{Xi}v_{Yi}$$

$$+ \frac{C}{2}\left(\sum_{\mathrm{X}}\sum_{\mathrm{i}} v_{Xi} - N\right)^2 +$$

$$+ \frac{D}{2}\sum_{\mathrm{X}}\sum_{\mathrm{Y,Y\neq X}}\sum_{\mathrm{i}} d_{XY} v_{Xi}\left(v_{Y,i+1} + v_{Y,i-1}\right)$$

X, Y – denote cities;

i,j – denote the tour stage

v_{Xi} – is the activation for each neuron, v_{Xi}=1 denotes the fact that city *X* is the i-th city visited in the tour

d_{XY} – denotes the distance between cities *X* and *Y*

The TSP Optimization Problem

Row constraint (first term) in the energy function is zero if and only if there is only one „1” in each order column; thus it takes care that no two or more cities are in the same travel order i.e. no two cities are visited simultaneously.

Column constraint (second term) is zero if and only if there is only one city appears in each order column; thus it takes care that each city is visited only once.

These terms represent the constraint of the problem.

The TSP Optimization Problem

Total number of „1” constraint (third term) is zero if and only if there are only N number of „1” appearing in the whole N*N matrix; thus is takes into care that all cities are visited.

That term represents the constraint of the problem.

The **fourth term** measures the tour length corresponding to a given tour, where the two terms inside the parenthesis stand for two neighboring visiting cities if V_{Xi} implying the tour length is calculated twice.

That term represents the objective function.

The TSP Optimization Problem

The fourth term mmeasures the total distance by adding intercity distances $d_{X,Y}$ for each pair of adjacent cities.

Here **A**, **B** and **C** are positive integers, the setting of these constants are critical for the performance of Hopfield network.

The TSP Optimization Problem

The set solving the TSP problem for N = 4.

The route: 3-2-4-1

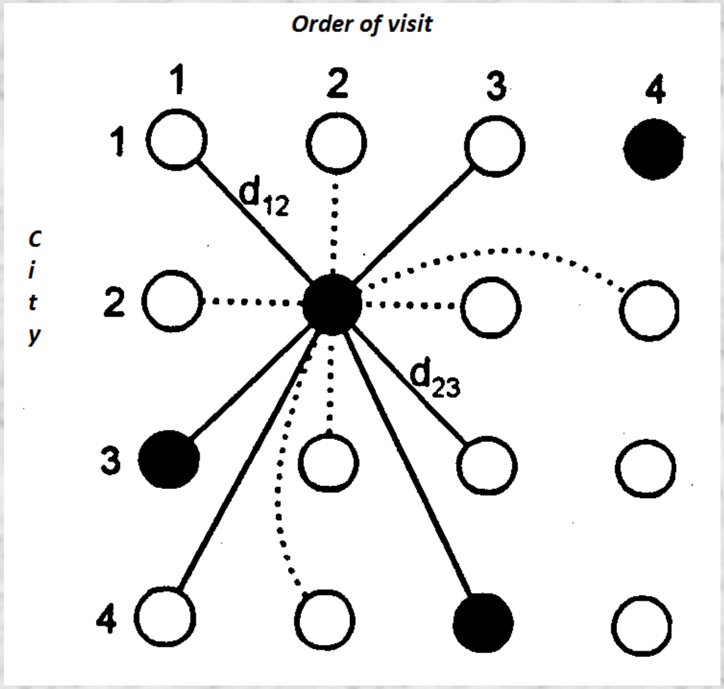
● tern on element

○ tern off element

connections for V_{22}

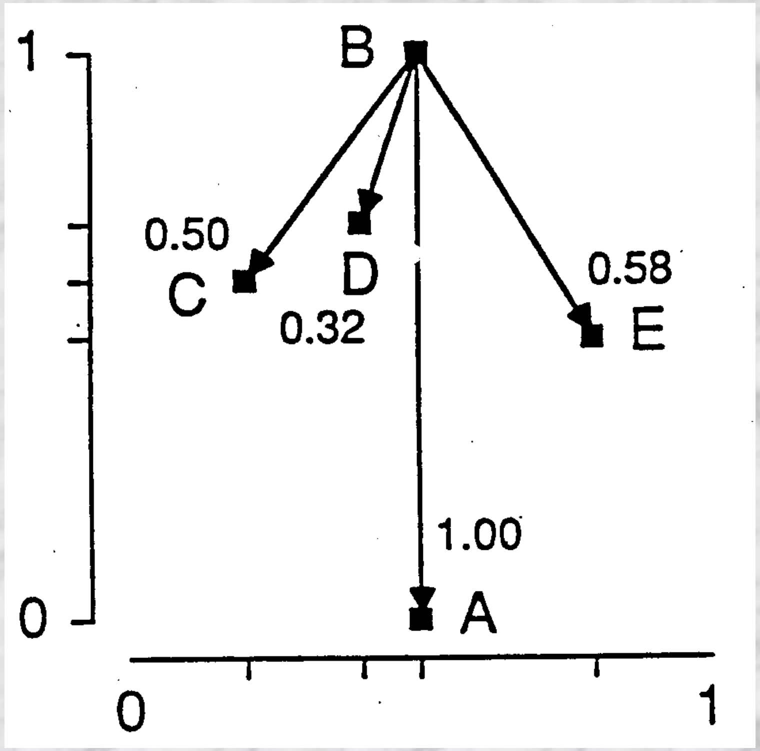
— inhibitory, weight d_{XY}

·····inhibitory, weight α



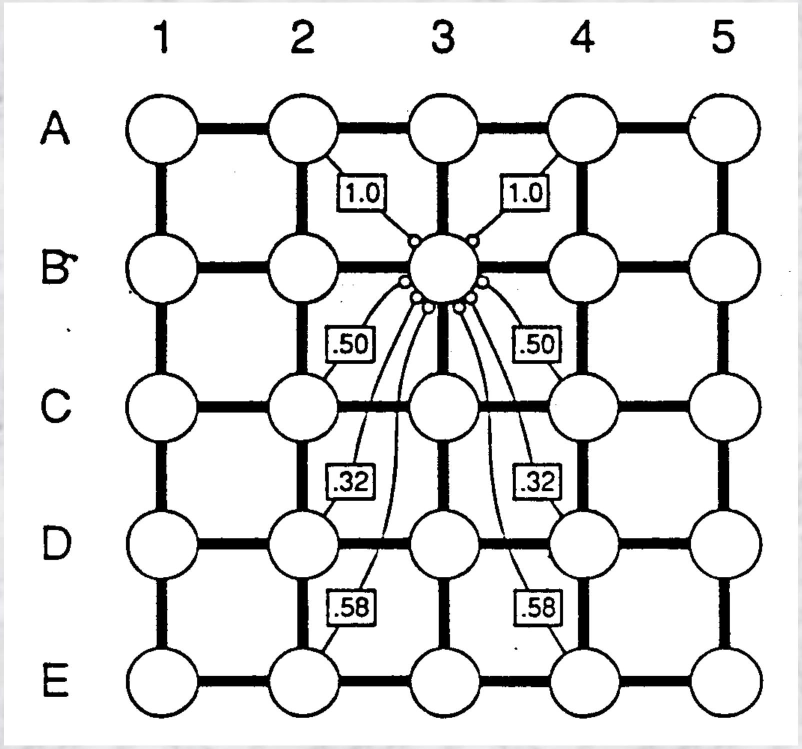
The TSP Optimization Problem

The example of normalized distances from the city B to other cities.



The TSP Optimization Problem

- Inhibitory connections of the V_{B3} element with the rest of cities



Neural Networks for Matrix Algebra Problems

Neural Networks for Matrix Algebra Problems

The feedforward neural networks for solving (in real time) a large variety of important matrix algebra problems such as:

- matrix inversion,
- matrix multiplication
- LU decomposition,
- the eigenvalue problem

Neural Networks for Matrix Algebra Problems

These algorithms basing on the massively parallel data transformation assure the high speed (µsek) in practice – in the real-time.

For a given problem define the error (energy) function and proper multilayer network and during learning phase find the minimum of the error function

Neural Networks for Matrix Algebra Problems

Matrix inversion

Let **A** be a nonsingular square

Goal:

To find the neural network calculating the matrix $\mathbf{B} = \mathbf{A}^{-1}$. matrix **B** fulfill the relation

$$\mathbf{BA} = \mathbf{I}$$

Neural Networks for Matrix Algebra Problems

Multiplying both sides by arbitrary non-zero vector **x**=[x_1,x_2,...,x_n] we get

$$\mathbf{BAx - x = 0} \qquad \textbf{(1)}$$

The energy (error) function can be defined by

$$E = \frac{1}{2}\left\| BAx - x \right\| \qquad (2)$$

Neural Networks for Matrix Algebra Problems

Solving of equation (1) can be replaced by the minimization of the function (2).

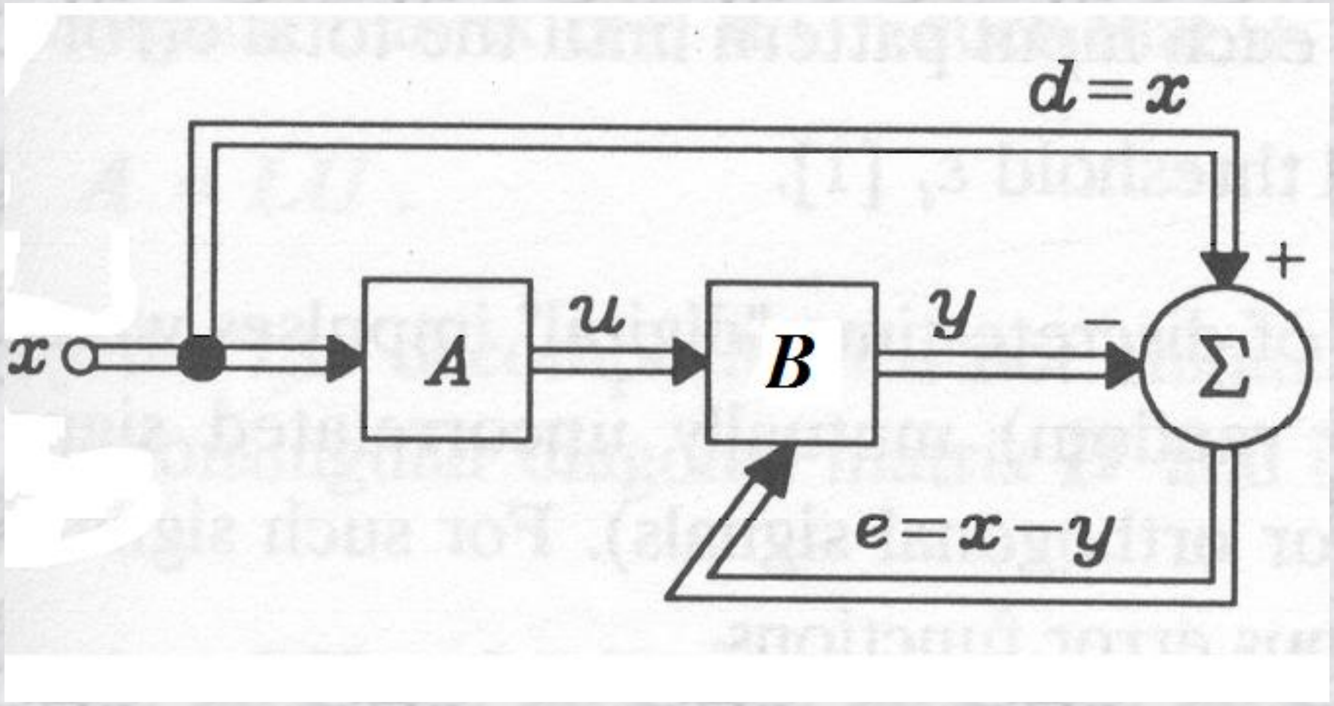
Vector **x** plays double role:

- is the learning signal (network input signal),
- is the desired output (target) signal

i. e. **it is the autoassociative network**

Neural Networks for Matrix Algebra Problems

A simplified block diagram



Neural Networks for Matrix Algebra Problems

$\mathbf{u} = \mathbf{Ax}, \quad \mathbf{y} = \mathbf{Bu}$ or

$\mathbf{y} = \mathbf{Bu} = \mathbf{BAx} = \mathbf{Ix} = \mathbf{x}$

It means that the output vector signal **y** must be equal to the input vector signal **x** – i.e. the network should learn the identity map **y** = **x.**

The fundamental question for the training phase:

what kind of input signals x should be applied in order to obtain the desired solution?

Neural Networks for Matrix Algebra Problems

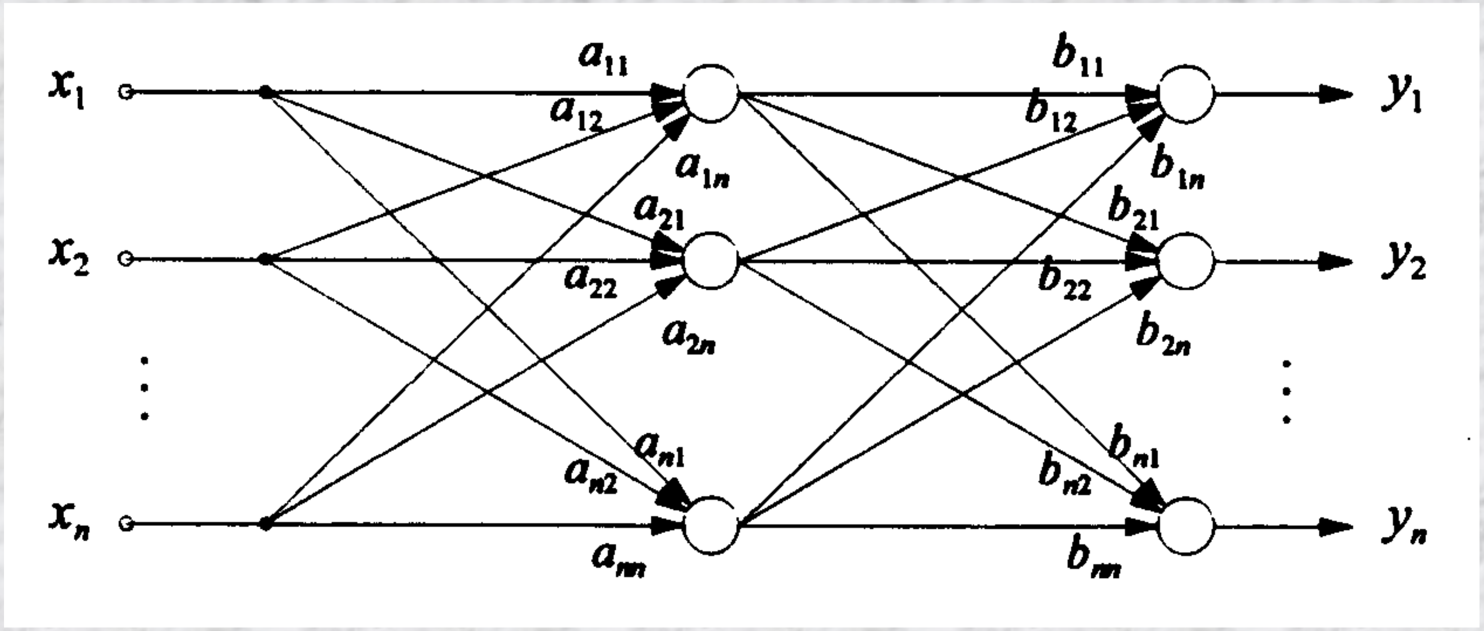
One of the simplest input patterns can be chosen as:
$\mathbf{x}^{(1)}=[1,0,0,...,0]^T$, $\mathbf{x}^{(2)}=[0,1,0,...,0]^T$,..., $\mathbf{x}^{(n)}=[0,0,0,...,1]^T$.

The better convergence speed can be obtained by changing the input patterns randomly on each time step from the set
$\mathbf{x}^{(1)}=[1,-1,...,-1]^T$, $\mathbf{x}^{(2)}=[-1,1,-1,...,-1]^T$,...,
$\mathbf{x}^{(n)}=[-1,-1,...,1]^T$.

In this two-layer network the first layer has fixed connection weights a_{ij}, while in the second layer weights are unknown, and are described by the unknown matrix $\mathbf{B} = \mathbf{A}^{1}$.

Neural Networks for Matrix Algebra Problems

The network architecture



Neural Networks for Matrix Algebra Problems

In order to minimize the local error function ***E***

$$E = \frac{1}{2}\sum_{j=1}^{n} e_j^2 = \frac{1}{2}\sum_{j=1}^{n}(x_j - y_j)^2$$

for a single pattern

y_i is the actual output signal
x_i is the desired output signal

we can apply a standard steepest-descent approach

$$\Delta B_{ij} = -\mu(y_i - x_i)u_j$$

Neural Networks for Matrix Algebra Problems

Matrix multiplication

If matrix **C** is equal the product of matrices **A** and **B** it fulfills the equation

$$\mathbf{C} = \mathbf{AB}$$

Neural Networks for Matrix Algebra Problems

To construct a proper neural network able to solve the problem it is necessary to define the error (energy) function whose minimization leads to the desired solution. Multiplying both sides by arbitrary non-zero vector $\mathbf{x}=[x_1,x_2,\ldots,x_n]$ we get

$$\mathbf{ABx} - \mathbf{Cx} = \mathbf{0}$$

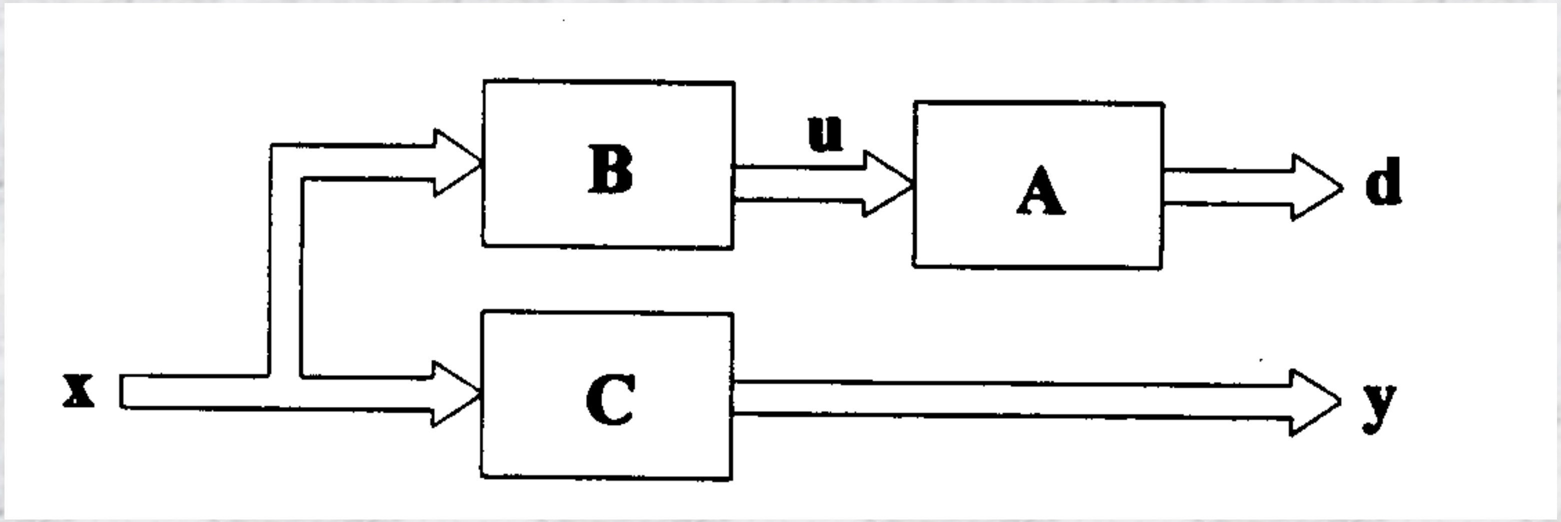
Neural Networks for Matrix Algebra Problems

On the basis of this equation we can define the error (energy) function

$$E = \frac{1}{2}\left(\left\|ABx - Cx\right\|_2\right)^2$$

Neural Networks for Matrix Algebra Problems

A simplified block diagram for matrix multiplication. In real it is one-layer network in spite that on the diagram there are three layers



Neural Networks for Matrix Algebra Problems

Only one out of these three layers responsible for matrix **C** is the subject of a learning procedure – realizing the equation

$$y = \mathbf{C}\mathbf{x}$$

After the learning process the network has to fulfill the equation **C** = **AB** in the diagram there are two additional layers with constant weights (the elements of matrices **A** and **B** respectively).

Neural Networks for Matrix Algebra Problems

These layers are used to compute the vector **d**, according to

$$\mathbf{d} = \mathbf{A}\mathbf{u} = \mathbf{A}\mathbf{B}\mathbf{x}$$

Again we can apply a standard steepest-descent algorithm. The adaptation rule has the form $c_{ij}(t+1) = c_{ij}(t) + \eta(d_{ij} - y_{ip})x_{jp}$

where *p* is the number of a learning pattern.

Neural Networks for Matrix Algebra Problems

LU decomposition

The standard **LU** decomposition of a square matrix **A** into: lower-triangular matrix **L** and upper-triangular matrix **U** such that:

$$\mathbf{A} = \mathbf{LU}$$

generally the **LU** decomposition is not unique. However, if the **LU** is factorization for a lower-triangular matrix **L** with unit diagonal elements factorization is unique.

Neural Networks for Matrix Algebra Problems

Multiplying both sides by arbitrary non-zero vector **x**=[x_1,x_2,...,x_n] and after some further transformation we get the energy function

$$E = \frac{1}{2}(\|LUx - Ax\|_2)^2$$

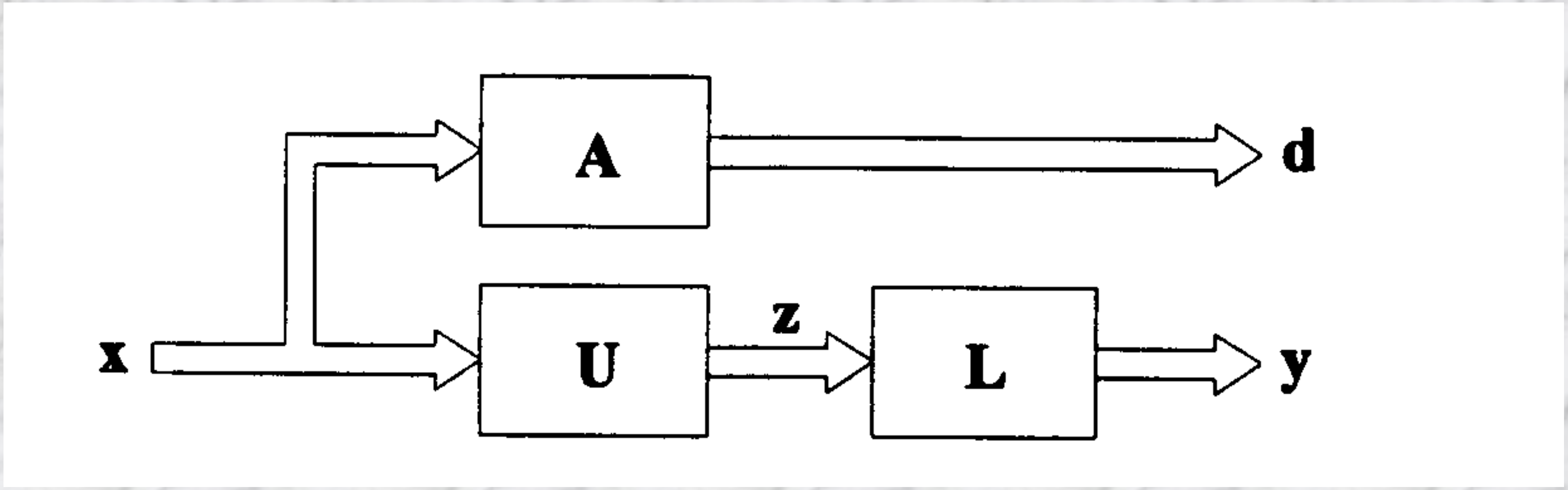
Neural Networks for Matrix Algebra Problems

The two-layer linear network is more complicated than the network for the matrix inversion or multiplication.

Here, both layers are the subject of learning procedure. The connection weights of the first layer are described by the matrix **U** and the second layer by the matrix **L**.

Neural Networks for Matrix Algebra Problems

A simplified block diagram



Neural Networks for Matrix Algebra Problems

The first layer performs a simple linear transformation **z** = **Ux**, where **x** is a given input vector. The second layer performs transformation **y = Lz = LUx.**

The parallel layer with weights defined by the matrix A elements is used to calculate the desired (target) output **d = Ax.**

Neural Networks for Matrix Algebra Problems

The weights l_{ii} are fixed and equal to unity, and proper elements of the matrices **L** and **U** are equal to zero.

To minimize the error function we will apply the simplified back-propagation algorithm.

Neural Networks for Matrix Algebra Problems

We get

$$l_{ij}(t+1) = l_{ij}(t) + \eta e_{ip} z_{ip}$$

for i > j, and

$$u_{ij}(t+1) = u_{ij}(t) + \eta \left[\sum_{h=1}^{n} l_{hi}(t+1) e_{hp} \right] x_{jp}$$

dla $i \leq j$

Neural Networks for Matrix Algebra Problems

where

$e_{ip} = d_{ip} - y_{ip}$ is the actual error of *i*-th output element for *p*-th pattern $\mathbf{x}_p$

$z_{ip} = \sum_{j=1}^{n} u_{ij} x_{jp}$ is the actual output of *i*-th element of the first layer for the same *p*-th pattern $\mathbf{x}_p$

and $y_{ip} = \sum_{j=1}^{i} l_{ij} z_{jp}$ $d_{ip} = \sum_{j=1}^{n} a_{ij} x_{jp}$