Neural Networks: Basic

Dokuz Eylül University
Computer Engineering Department

24 October 2001 –

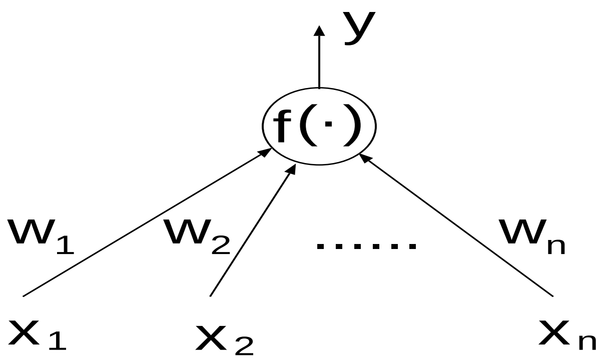
□ What is Neural Network?

- A system with *neurons* connected with *synapses*,

  where

  - each synapse:
    - is given a value called *weight*;
    - transmits value with being multiplied by its weight,
  - each neuron:
    - is given a function called *transfer function*;
    - takes multiple (fan-in) inputs, caluculates their weighted sum and outputs its function value.

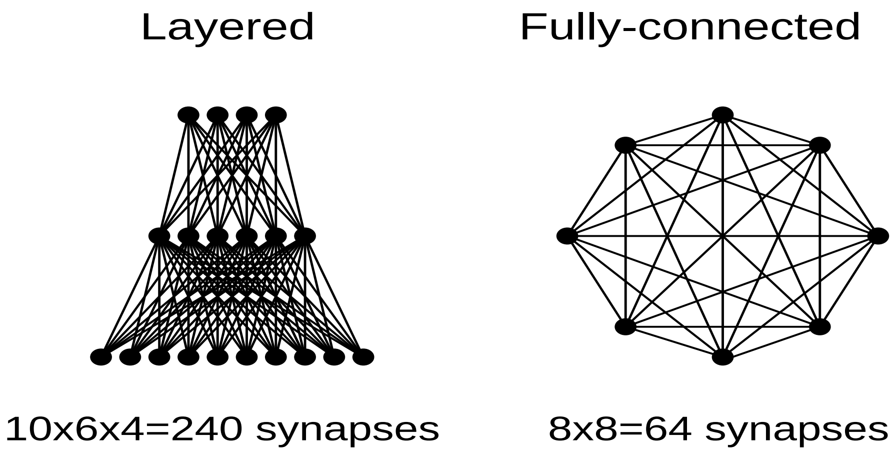
  that is,



$$y = f(\sum_{i=0}^{n} w_i \cdot x_i). \tag{1}$$

- Typically categorized into two forms:
  1. Layered (feed forward);
  2. Fully-connected

For example:



□ **Neural Networks What for?**

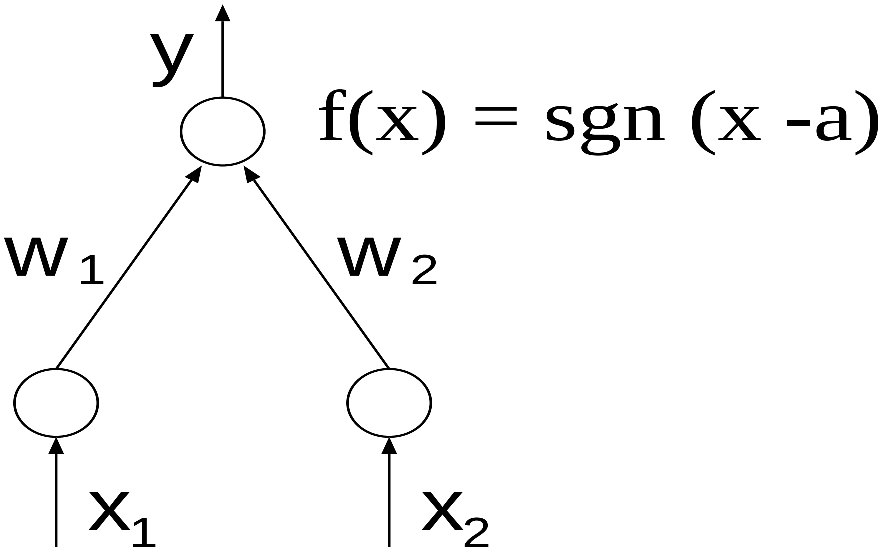
- To model human brain;
- To make computers try things that we are good at but they are not. [1]

[1]Terrence J. Sejnowski said in his foreword for the book "Face Image Analysis by Unsupervised Learning" by Marian Stewart Bartlett. Kluwer Academic Publishers (2001) that *Computers are good at many things that we are not good at, like sorting a long list of numbers and calculating the trajectory of a rocket, but they are not at all good at things that we do easily and without much thought, like seeing and hearing.*

□ **The simplest example.**

Excersise 1 *Specify three parameters w_1, w_2 and a so that a neural network with two inputs and one output can caluculate the following three operations.*

1. *AND*
2. *OR*
3. *XOR*



Did you notice that XOR cannot be rearized with this framework?

$\Downarrow$

Then try it by adding one additional layer.

Excersise 2 *Why do you think XOR cannot be realized in this way? Plot the four points and try to separate them with a line*

$$w_1x_1 + w_2x_2 = a.$$

□ Perceptron

- A two-layered neural network.

  Suppose:

  - M inputs: $x_1, \cdots x_M$;
  - N outputs: $y_1, \cdots y_N$;
  - each N output neurons are connected with all M input neurons;
  - weight values from the j-th input neuron to the i-th output neuron is w_{ij};

  then

$$y_i = f(\sum_{j=1}^{M} w_{ij} x_j), \tag{2}$$

  where, typically,

$$f(x) = \mathrm{sgn}(x - a). \tag{3}$$

Excersise 3 *Implement a perceptron which has arbitrary number of inputs and outputs, with randomly chosen weight values and with transfer function of all neurons being sgn(·) such that the perceptron reads p input data and outputs p results of the caluculation.*

□ Perceptron Learning

— How can we determine weights?

- Assume M input and N output neurons.

- Also assume that we have p examples

$$x_j^\mu \quad (\mu = 1, 2, \cdots, p, \quad j = 1, 2, \cdots, M) \tag{4}$$

whose desired output are given, i.e.,

$$z_i^\mu \;\; (i = 1, 2, \cdots, N), \tag{5}$$

while real outputs are

$$y_i^\mu \quad (i = 1, 2, \cdots, N). \tag{6}$$

- Then we modify w_{ij} by adding:

$$\delta w_{ij} = \epsilon (z_i^\mu - y_i^\mu) \cdot x_j^\mu, \tag{7}$$

in each iteration of giving example, and we obtain

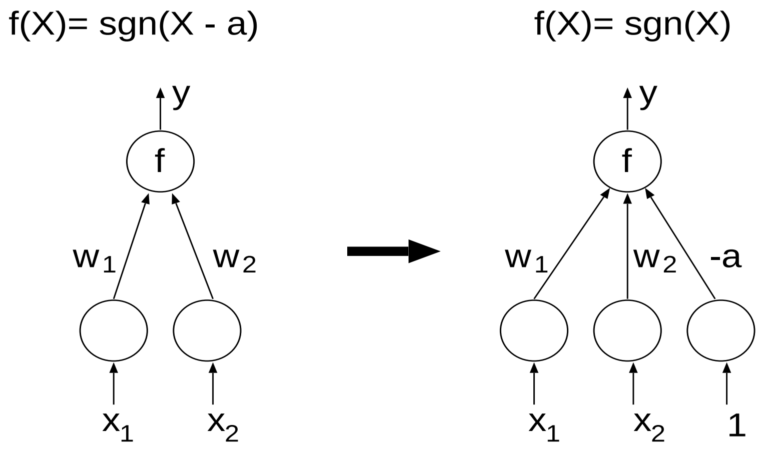
$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \delta w_{ij} \tag{8}$$

(implying *if correct do nothing, otherwise modify.)*

- Note that each of $x_k^\mu,\;\, y_k^\mu,\;\, and\;\, z_k^\mu$ takes either 1 or -1.

□ **How to determine threshold?**

also automatically.



In both cases, we can obtain same output y, that is,

$$y = \text{sgn}(w_1 \cdot x_1 + w_1 \cdot x_1 - a).$$

So, we can also specify a as one of weights.

Excersise 4 *Explore Perceptron Learning in the following two ways.*

1. *Obtain two weights and threshold value to solve two inputs and one output perceptron by a manual calculation.*
2. *Assume you have 100 points each from 2 different categories of regions in the 2-dimensional x-y coordinate. Design a neural network which classifies a given point to either of this two categories.*

□Analog Neurons

- accept analog inputs and output analog values;
- with its transfer function usually being: [2]

$$f(X) = \tanh(\beta X) \tag{16}$$

Excersise 5 *Answer the following questions.*

(1) Draw the graph of the function:

$$y = tanh(\beta x)$$

with three diferent value of $\beta = 0.1,\ \ 1$ *and* 10*.*

(2) When $f(x) = \tanh(\beta x)$ *verify the relation*

$$f'(x) = \beta\{1 - f^2(x)\}$$

and think what it means both in the case when x *is large enough and* $x = 0$*.*

---

[2] From the definitions:

$$e^{jx} = \cos x + j \sin x; \tag{9}$$

$$e^{-jx} = \cos x - j \sin x, \tag{10}$$

we obtain

$$\cos x = \frac{e^{jx} + e^{-jx}}{2}; \tag{11}$$

$$\sin x = \frac{e^{jx} - e^{-jx}}{2j}; \tag{12}$$

We extend them to

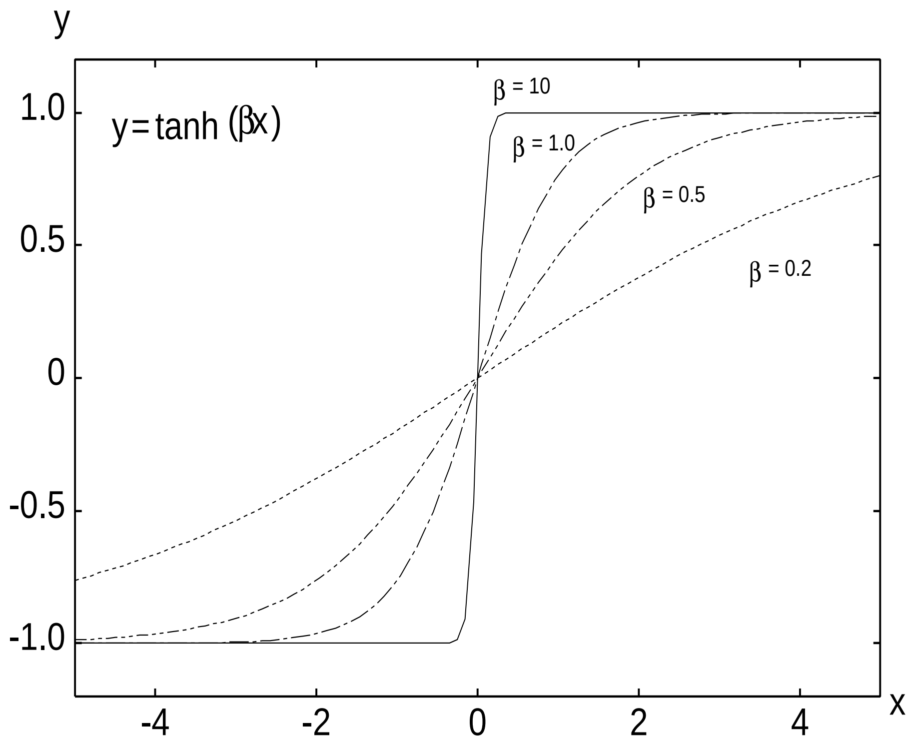
$$\cosh x = \frac{e^{x} + e^{-x}}{2}; \tag{13}$$

$$\sinh x = \frac{e^{x} - e^{-x}}{2}; \tag{14}$$

So,

$$\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{15}$$

- $y = \tanh(\beta x)$



□ Gradient Learning

— Learning for analog neurons.

- Assume that we have M inputs N outputs, and p examples.
- All neurons are analog.
- *Total Squared Error* of outputs is difened as:

$$D = \sum_{\mu=1}^{p} \sum_{i=1}^{N} (y_i^\mu - z_i^\mu)^2 \tag{17}$$

where

$$y_i^\mu = f(\sum_k w_{ik} x_k^\mu) \tag{18}$$

and usually

$$f(h) = \tanh(\beta h). \tag{19}$$

- The Equation (24) defines an *error surface* of high dimensionality.
- Then, the *gradient* of D with respect to w_{ij} modifies w_{ij} as:[3]

$$\delta w_{ik} \equiv -\epsilon \frac{\partial D}{\partial w_{ik}} = \cdots = \epsilon \sum_\mu (\zeta_i^\mu - S_i^\mu) f'(\sum_k w_{ik} \sigma_k^\mu), \tag{21}$$

where

$$f'(x) = \beta\{1 - f(x)^2\}. \tag{22}$$

[3]Though there exist many alternatives to find minima of a non-linear function $g(\mathbf{x})$, *steepest descent*

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \epsilon \nabla g(\mathbf{x}_n) \tag{20}$$

is sufficient in most cases.

- A toy example:

Excersise 6 *Assuming a training set ($p = 8$) bellow, answer the following questions.*

μ	x_1	x_2	z_1
1	3.2	-2.4	0.8
2	-4.8	-3.5	0.6
3	-2.7	3.5	0.4
4	5.3	3.5	0.2
5	-1.9	3.5	0.4
6	6.2	-3.5	0.8
7	7.6	3.5	0.2
8	-3.3	-3.5	0.6

(1) What kind of task do you imagine with the training set.

(2) Guess the output when $\mathbf{x} = (6.8, -7.4)$ *is given as an input.*

(3) Train a neural network with $M = 2$ *and* $N = 1$ *using Gradient Learning.*

□ Multi-layered Neural Networks

- Separability
  - If we can separate the solution space by hyperplane,

    (To be more specific, we can separate 2D solution space of 2-input AND/OR by a straight line, but not for XOR.)

    then we can solve the problem by single perceptron.
  - Multi-layer allows us to separate the space by more flexible surface, not necessarily a hyper-plane.

- Definition of variables:
  - L layers of neurons.
  - k_r neurons in the r-th layer. $\quad r = 1, 2, \cdots, L$.
  - M input neurons ($k_1 = M$) and N output neurons ($k_L = N$).
  - w_{ij}^r: weight from the j-th neuron in the r-th layer to the i-th neuron in the r-th layer.
  - p training pairs: $(\mathbf{x}(\mu), \mathbf{z}(\mu))$, $\quad i = 1, 2, \cdots, p$.
  - $y_i^r(\mu)$: the output of the i-th neuron in the r-th layer when the μ-th training sample is given.
  - fan-in's to j-th neuron in the r-th layer when the μ-th training sample is given.

    $$h_j^r(\mu) \equiv \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) + w_{j0}^r = \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i), \tag{23}$$

    where

    $$y_0^{r-1} = 1$$

    and

    $$(y_1^1(\mu), \cdots, y_M^1(\mu)) = (x_1(\mu), \cdots, x_M(\mu)) = \mathbf{x}(\mu).$$
  - Each output neuron's error:

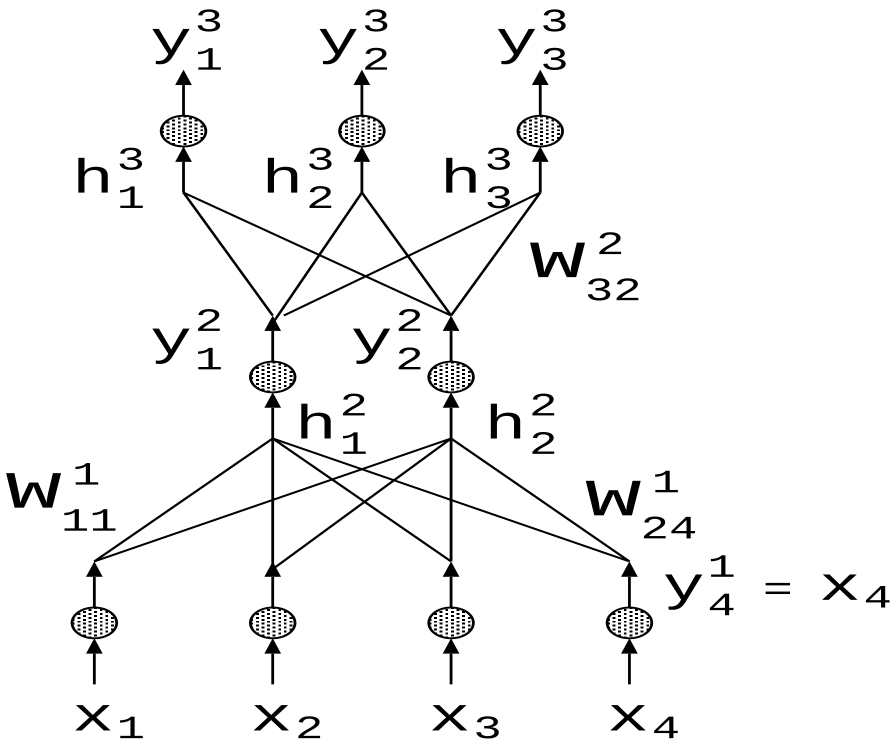
    $$e_i(\mu) \equiv y_i(\mu) - z_i(\mu), \;\; i = 1, 2, \cdots, N. \tag{24}$$

    where $(z_1(\mu), \cdots, z_N(\mu)) = \mathbf{z}(\mu)$
  - Error function to be minimized:

    $$D = \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i=1}^{N} e_i(\mu)^2 \tag{25}$$

- An example of the notations

  - $L = 3$, $M = k_1 = 4$, $N = k_3 = 3$ and $k_2 = 2$.



  - e.g.

    ⋆ $h_2^2(\mu) = w_{21}^1 \cdot x_1(\mu) + w_{22}^1 \cdot x_2(\mu) + w_{23}^3 \cdot x_3(\mu) + w_{24}^1 \cdot x_4(\mu)$.

    ⋆ $h_2^3(\mu) = w_{21}^2 \cdot y_1^2(\mu) + w_{22}^2 \cdot y_2^2(\mu)$.

□ Back Propagation

— Learning for multi-layered analog neural networks.

- The algorithm:

1. *Initialization:*

   · Initialize all the weights with small random values.

2. *Forward computations:*

   · For each of the training vectors

$$\mathbf{x}(i), \quad \mu = 1, 2, \cdots, p$$

   compute all the

$$y_j^r(\mu) = f(h_j^r(\mu)) \tag{26}$$

   for $j = 1, 2, \cdots, k_r$ and $r = 1, 2, \cdots, L$.

3. *Backward computations:*

   · For each $\mu = 1, 2, \cdots, p$ and $j = 1, 2, \cdots, N$ compute

$$\delta_j^L(\mu) = e_j(\mu) f'(h_j^L(\mu)). \tag{27}$$

   · Then compute

$$\delta_j^{r-1}(\mu) = e_j^{r-1}(\mu) f'(h_j^{r-1}(\mu)) \tag{28}$$

   for $r = L, L-1, \cdots, 2$ and $j = 1, 2, \cdots, k_r$ where

$$e_j^{r-1}(\mu) = \sum_{k=1}^{k_r} \delta_k^r(\mu) w_{kj}^r. \tag{29}$$

4. Update the weights for $r = 1, 2, \cdots, L$ and $j = 1, 2, \cdots, k_r$

$$w_{jk}^r\,[\text{new}] = w_{jk}^r\,[\text{old}] + \triangle w_{jk}^r \tag{30}$$

   where

$$\triangle w_{jk}^r = -\epsilon \delta_j^r(\mu) y_k^r(\mu). \tag{31}$$

   until no change occurs during the training cycle of $\mu = 1, 2, \cdots, p$.

($\star$ Forward calculation in our previous toy example: From the bottom to top.)

$$y_1^1(\mu) = x_1(\mu), \quad y_2^1(\mu) = x_2(\mu), \quad y_3^1(\mu) = x_3(\mu), \quad y_4^1(\mu) = x_4(\mu).$$

$$\Downarrow$$

$$h_1^2(\mu) = w_{11}^1 \cdot y_1^1 + w_{12}^1 \cdot y_2^1 + w_{13}^1 \cdot y_3^1 + w_{14}^1 \cdot y_4^1 \qquad \text{from (23)}$$

$$\Downarrow$$

$$\text{similarly } h_2^2(\mu).$$

$$\Downarrow$$

$$y_1^2 = f(h_1^2(\mu)) \qquad \text{from (26)}$$

$$\Downarrow$$

$$\text{similarly } y_2^2(\mu).$$

$$\Downarrow$$

$$h_1^3(\mu) = w_{11}^2 \cdot y_1^2 + w_{12}^2 \cdot y_2^2 \qquad \text{from (23) again}$$

$$\Downarrow$$

$$\text{similarly } h_2^3(\mu) \quad \text{and} \quad h_3^3(\mu).$$

$$\Downarrow$$

$$y_1^3 = f(h_1^3(\mu)) \qquad \text{from (26) again}$$

$$\Downarrow$$

$$\text{similarly } y_2^3(\mu) \quad \text{and} \quad y_3^3(\mu).$$

(⋆ Backward calculation in our previous toy example: From the top to bottom.)

$$e_1^{(3)} = y_1^3 - z_1(\mu) \qquad \text{from (24)}$$

$$\Downarrow$$

$$\text{similarly } e_2^{(3)}(\mu) \qquad \text{and} \qquad e_3^{(3)}(\mu).$$

$$\Downarrow$$

$$\delta_1^3(\mu) = e_1^{(3)} \cdot f'(h_1^3(\mu)) \qquad \text{from (66)}$$

$$\Downarrow$$

$$\text{similarly } \delta_2^3(\mu) \qquad \text{and} \qquad \delta_3^3(\mu).$$

$$\Downarrow$$

$$e_1^2(\mu) = \delta_1^3(\mu) \cdot w_{11}^2 + \delta_2^3(\mu) \cdot w_{21}^2 + \delta_3^3(\mu) \cdot w_{31}^2$$

$$\Downarrow$$

$$\text{similarly } e_2^2(\mu) \qquad \text{from (29).}$$

$$\Downarrow$$

$$\delta_1^2(\mu) = e_1^2(\mu) \cdot f'(h_1^2(\mu)) \qquad \text{from (28)}$$

$$\Downarrow$$

$$\text{similarly } \delta_2^2(\mu).$$

$$\Downarrow$$

$$\begin{gathered} e_1^1(\mu) = \delta_1^2(\mu) \cdot w_{11}^1 + \delta_2^2(\mu) \cdot w_{21}^1 \\ e_2^1(\mu) = \delta_1^2(\mu) \cdot w_{12}^1 + \delta_2^2(\mu) \cdot w_{22}^1 \\ e_3^1(\mu) = \delta_1^2(\mu) \cdot w_{13}^1 + \delta_2^2(\mu) \cdot w_{23}^1 \\ e_4^1(\mu) = \delta_1^2(\mu) \cdot w_{14}^1 + \delta_2^2(\mu) \cdot w_{24}^1 \\ \text{from (29) again} \end{gathered}$$

$$\Downarrow$$

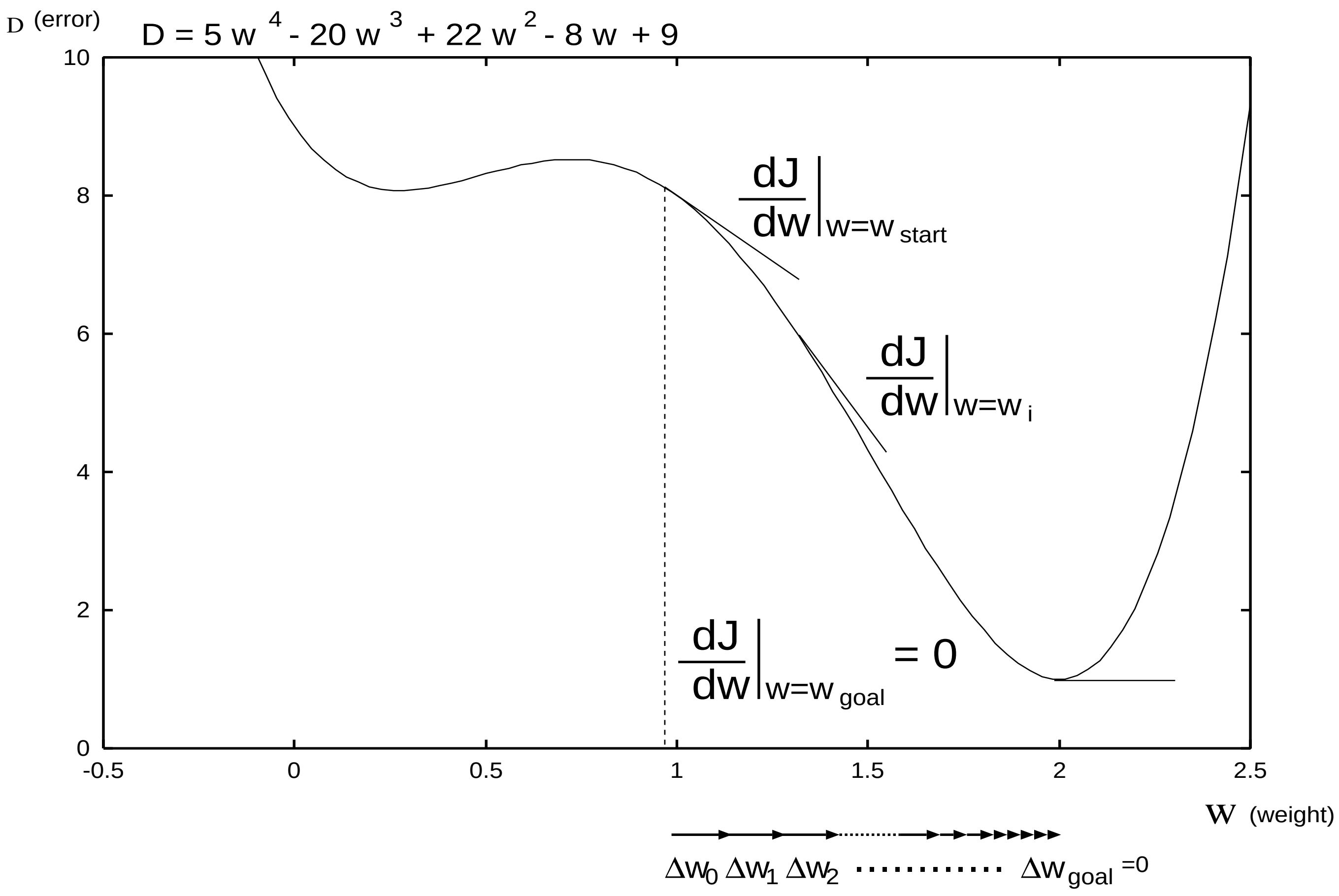
$$\delta_1^1(\mu) = e_1^1(\mu) \cdot f'(h_1^1(\mu)) \qquad \text{from (28) again}$$

$$\Downarrow$$

$$\text{similarly } \delta_2^1(\mu), \quad \delta_3^1(\mu), \quad \delta_4^1(\mu).$$

• Rationale

⋆ Thought Experiment: 1-D weight



Modification:

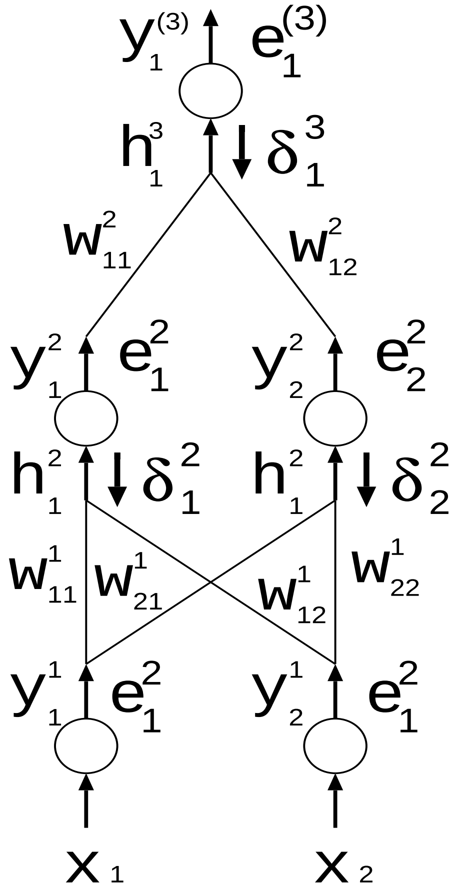
$$w^{\text{new}} = w^{\text{old}} + \triangle w \tag{32}$$

where

$$\triangle w = -\epsilon \cdot \frac{dD}{dw}|_{w=w^{\text{old}}} \tag{33}$$

means that starting from w_0, w gradually approaches the point where D takes a minimum value with δw being decreased.

⋆ An example: XOR (revisited)



In this example,

$$
\begin{aligned}
D &= \frac{1}{2}\{e_1^{(3)}\}^2 \\
&= \frac{1}{2}\{y_i^{(3)} - z_i\}^2 \\
&= \frac{1}{2}\{(f(h_i^3) - z_i\}^2 \\
&= \frac{1}{2}\{f(w_{11}^2 \cdot y_1^2 + w_{12}^2 \cdot y_2^2) - z_i\}^2 \\
&= \frac{1}{2}\{f(w_{11}^2 \cdot f(h_1^2) + w_{12}^2 \cdot f(h_2^2) - z_i\}^2 \\
&= \frac{1}{2}\{f(w_{11}^2 \cdot f(w_{11}^1 \cdot y_1^1 + w_{12}^1 \cdot y_2^1) + w_{12}^2 \cdot f(w_{21}^1 \cdot y_1^1 + w_{22}^1 \cdot y_2^1) - z_i\}^2 \\
&= \frac{1}{2}\{f(w_{11}^2 \cdot f(w_{11}^1 \cdot x_1 + w_{12}^1 \cdot x_2) + w_{12}^2 \cdot f(w_{21}^1 \cdot x_1 + w_{22}^1 \cdot x_2) - z_i\}^2
\end{aligned}
$$

You can see that D is a function of only all the weights (6-D in this case).

⋆ Is D a function of all the weights?

We have studied that

$$\begin{aligned} D &= \frac{1}{2}\sum_{i=1}^{N} {e_i}^2 && (34) \\ &= \frac{1}{2}\sum_{i=1}^{N}(y_i - z_i)^2 && (35) \\ &= \frac{1}{2}\sum_{i=1}^{N}(f(h_i^L) - z_i)^2 && (36) \end{aligned}$$

with

$$h_i^{r+1} = \sum_{j=1}^{k_r} w_{ij}^r \cdot y_j^r \tag{37}$$

and

$$y_j^r = f(h_j^r). \tag{38}$$

Hence we can conclude that D is a function of all the w_{ij}^r as in the example of XOR shown in the previous page.

In order to obtain $\triangle w_{ij}^r$, we calculate

$$\frac{\partial D}{\partial w_{ij}^r} = \frac{\partial D}{\partial h_i^{r+1}} \cdot \frac{\partial h_i^{r+1}}{\partial w_{ij}^r} \tag{39}$$

where we know from (37)

$$\frac{\partial h_i^{r+1}}{\partial w_{ij}^r} = y_j^r \tag{40}$$

What is to be done here is to put

$$\delta_i^{r+1} \equiv \frac{\partial D}{\partial h_i^{r+1}} \tag{41}$$

⋆ Where are the definitions of δ_j^{r-1} and e_j^{r-1} in (28) and (29) from?

Again starting with

$$D = \frac{1}{2}\sum_{j=1}^{N} {e_j}^2 = \frac{1}{2}\sum_{j=1}^{N}(y_j - z_j)^2 = \frac{1}{2}\sum_{j=1}^{N}(f(h_j^L) - z_j)^2 \tag{42}$$

So when $r = L$

$$\delta_j^L \equiv \frac{\partial D}{\partial h_j^L} = e_j^L \cdot f'(h_j^L). \tag{43}$$

When $r < L$ holds D can be expressed as a function of h_k^r for

$$k = 1, 2, \cdots, k_r$$

and each of these h_k^r are influenced by h_j^{r-1}. Hence δ_j^{r-1} is defined as [4]

$$\delta_j^{r-1} \equiv \frac{\partial D}{\partial h_j^{r-1}} = \sum_{k=1}^{k_r} \frac{\partial D}{\partial h_k^r} \cdot \frac{\partial h_k^r}{\partial h_j^{r-1}} = \sum_{k=1}^{k_r} \delta_k^r \cdot \frac{\partial h_k^r}{\partial h_j^{r-1}} \tag{46}$$

Here, since

$$h_k^r = \sum_{m=1}^{k_r} w_{km}^r \cdot y_m^r = \sum_{m=1}^{k_r} w_{km}^r \cdot f(h_m^{r-1}) \tag{47}$$

We obtain

$$\frac{\partial h_k^r}{\partial h_j^{r-1}} = \frac{\partial\left(\sum_{m=1}^{k_{r-1}} w_{km}^r \cdot f(h_m^{r-1})\right)}{\partial h_j^{r-1}} = w_{kj}^r \cdot f'(h_j^{r-1}) \tag{48}$$

Hence,

$$\delta_j^{r-1} = \sum_{k=1}^{k_r} \delta_k^r \cdot w_{kj}^r \cdot f'(h_j^{r-1}). \tag{49}$$

In conclusion, if we define

$$e_j^{r-1} = \sum_{k=1}^{k_r} \delta_k^r \cdot w_{kj}^r \tag{50}$$

we obtain

$$\delta_j^{r-1} = e_j^{r-1} \cdot f'(h_j^{r-1}) \tag{51}$$

[4] For a function of n variables each of which is also a function of t, that is,

$$F(x_1(t), x_2(t), \cdots, x_n(t)), \tag{44}$$

the partial derivative of F is

$$\frac{\partial F}{\partial t} = \frac{\partial F}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial F}{\partial x_2}\frac{\partial x_2}{\partial t} + \cdots + \frac{\partial F}{\partial x_n}\frac{\partial x_n}{\partial t}. \tag{45}$$

□ Learning by an Evolution:

- Artificial Evolution — From a biological analogy, the followings are defined.

Chromosome:
Each chromosome is comprised of *genes* each of which is occupied by an *allele*. It is also called an *individual.*

Population:
A number of individuals constructs a population.

Fitness:
The ability of each individual is evaluated as the degree to how the individual fits to the environment which is called a fitness value.

Selection:
Parents are selected to produce one offspring. The higher the fitness the more likely to be selected, e.g., (1) selection is always from the best half of the population, or (2) selection is made so that the probability is proportional to the fitness value.

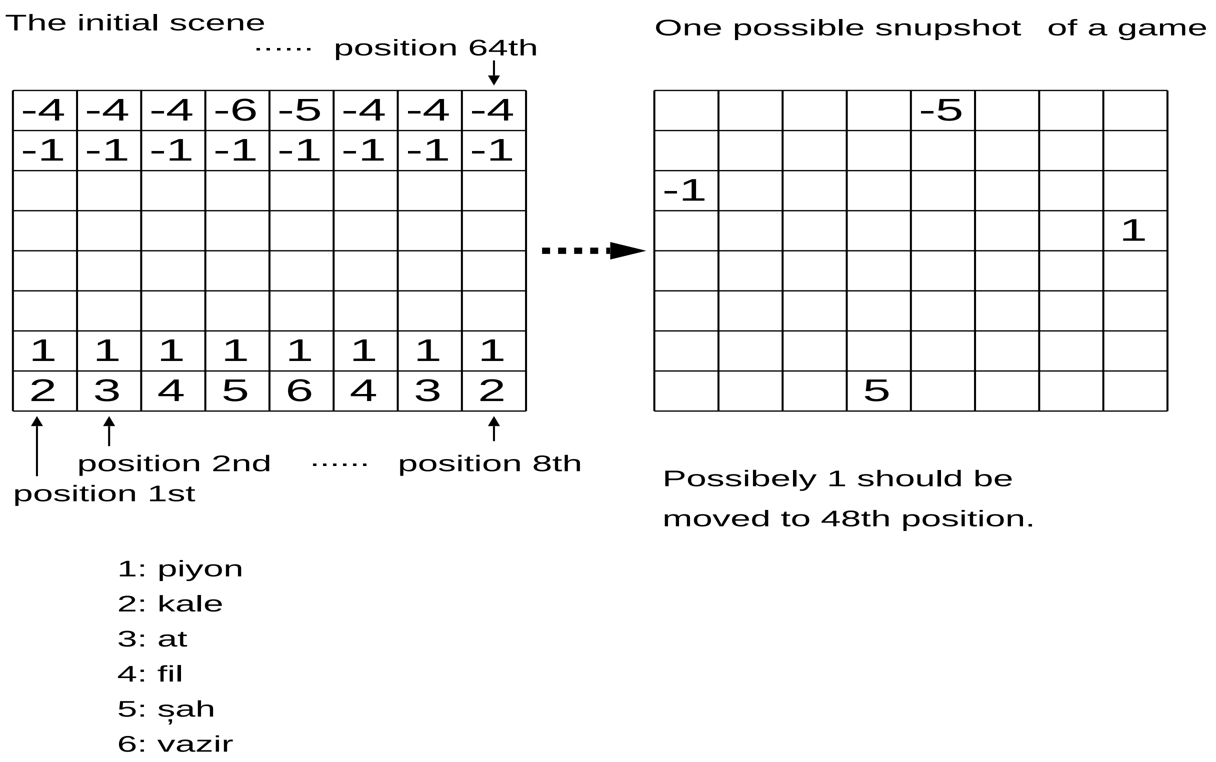
Crossover:
Child chromosome are produced by picking up genes either from those of the parent chromosomes at the same position.

Mutation:
To introduce new allele some of the genes are replaced with randomely generated new gene.

- Algorithm:

1. *Initialization:* Initialize all the individuals in the 1st generation at random.
2. *Fitness Evaluation:* Evaluate fitness of all the individuals of the population.
3. *Reproduction:*
   (1) Select two parents according to the fitness.
   (2) Recombine the parents to create an offspring.
   (3) Mutate the offspring.
   (4) Repeat (1) to (3) until new generation of same size is obtained.
4. *Evolution:* Repeat 2. to 3. until the perfect individual appears, or prespecified generations is reached.

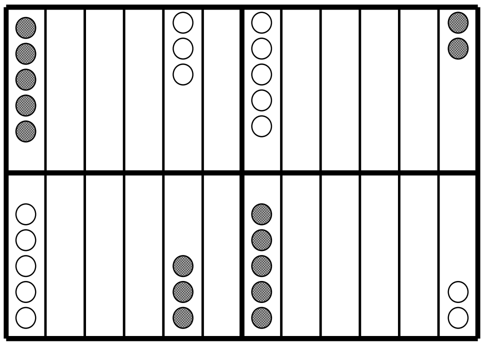
□ Chess (Satranç) by Neural Networks



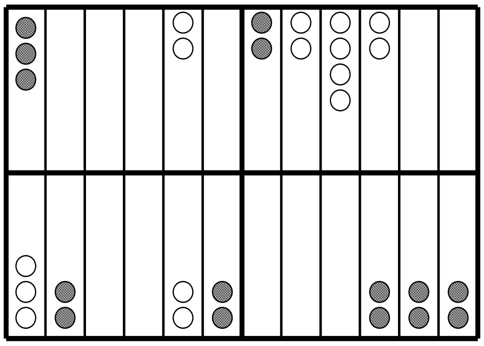
- One possible implementation:
  - 64 input neurons each of which takes a value from –6 to 6 representing a stone, plus one input binary neuron representing whose turn, which corresponds to one scene of a game.
  - 2 output neurons: one takes a value from –6 to 6 and the other from 1 to 64, representing a possible best movement, that is, which stone should be moved to where? (e.g., output 1 and 48 means that the stone-1 should be moved to

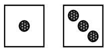
□ Backgammon (Tavla) by Neural Networks

The initial scene.



A possible one scene in a game.



When  what is the possible best movements?

- One possible implementation:
  - a total of 29 input neurons;
    - ⋆ 26 input neurons[5] each of which takes a value from -15 to 15, representing a number of stones at each cite, positive ones are mine and negative ones are opponent's, plus two neurons with each indicating the figure of two dice, and one more neuron representing whose turn.

- 4 output neurons[6]
  - ⋆ each of which takes a value from 1 to 6 which indicates how many positions the corresponding stone proceeds.

or

- 24 output neurons
  - ⋆ indicating the next scene as the 24 input neurons for the purpose above.

[5]two of which are for the outside position of mine and opponent's, respectively

[6]Usually we need two but if the figures of two dice are identical, the number of movement is four.

□ Reward and Penalty Learning:

An example on perceptron with one hidden layer.

- N_0 *analog* and *deterministic* output neurons:

$$y_i^2 = f(\sum_j w_{ij}^2 y_j^1) \tag{52}$$

  where

$$f(x) = \frac{1}{1+\exp(-x)} \tag{53}$$

- *Binary* and *stochastic* hidden neurons:

$$\begin{aligned} \mathrm{prob}(y_i^1 = 1) &= f(\sum_j w_{ij}^1 x_j) && (54) \\ \mathrm{prob}(y_i^1 = 0) &= 1 - f(\sum_j w_{ij}^1 x_j) && (55) \end{aligned}$$

- Using

$$D = \frac{1}{N_0} \sum_i (\hat{y}_i^2 - y_i^2)^2 \tag{56}$$

  – modify weights to output layer by *gradient* method as

$$\delta w_{ij}^{\mathrm{out}} = \epsilon(\hat{z}_i - z_i) f'(\sum_k w_{ik}^{\mathrm{out}} y_k) y_j \tag{57}$$

  – and weights to hidden layer in a stochastic way as

$$\begin{aligned} \delta w_{ij}^{\mathrm{hidden}} &= r\epsilon(y_i - f(\sum_k w_{ik}^{\mathrm{hidden}} x_k) x_j \\ &+ (1-r)\lambda\epsilon(1 - y_i - f(\sum_k w_{ik}^{\mathrm{hidden}} x_k) x_j && (58) \end{aligned}$$

  where r is signal meaning success/failure of the task, i.e.,

  ⋆ $r = 1$ with probability $1 - D$;

  ⋆ $r = 0$ with probability D.

  IF $rand() < 1 - D$;
  THEN $r = 1$;
  ELSE $r = 0$;

- Parameter notations of the previous few pages:

The i-th output neuron outputs z_i
while desirerd output is $\hat{z}_i$.

$$\Uparrow$$

The i-th output neuron takes

$$\sum_{k}^{N_{\text{hidden}}} w_{ik}^{\text{out}} y_k$$

from all the hidden neurons.

$$\Uparrow$$

Each of hidden neuron outputs y_k,

$$\Uparrow$$

The i-th hidden neuron takes

$$\sum_{k}^{N_{\text{input}}} w_{ik}^{\text{hidden}} x_k$$

from the input layer.

$$\Uparrow$$

Each input neuron takes an input x_k, and
send to all neurons in the hidden layer.

- Apply this to a task of symmetry detection.
  - n^2 input, 12 hidden neurons, and 3 output.
  - if the $n \times n$ pixels of the input are
    - horizontally symmetric
    - vertically symmetric
    - diagonally symmetric

    the 1st, 2nd and 3rd neuron should be acctivated.

- Interpretation of the reward and penalty learning.

□ Unsupervised Learning:

Unsupervised learning can only do anything useful when there is redundancy in the input data. *In this sence redundancy provides knowledge.*

- Learning by Competition.
- Kohonen's Self-organizing Map.
- Reinforcement Learning.

(1) Unsupervised Learning through Competition:

Let $\mathbf{x} = (x_1, ..., x_n)$ be a normalized input vector, i.e.,

$$\sum_{i=1}^{n} x_i = 1, \tag{59}$$

in n-dimensional space which we want to classify in one of k different clasters.

- The network consitsts of n real-valued input neurons and k binary output neurons (with all neurons zero-threshold.)
- Each output neuron receives
$$h_i = \sum_{j=1}^{n} w_{ij} x_j \tag{60}$$
from all the neurons in the input layer.

- Then the only neuron which recieves
$$h_m = \max_i (h_i) \tag{61}$$
will be on.

- Learning Algorithm:

1. Initialize all the weights w_{ij} at random, and normaliz them such that

$$\sum_{j=1}^{n} w_{ij} = 1 \quad \text{for} \quad \forall i = 1, ..., k \tag{62}$$

2. Selct a vector $\mathbf{x}^{\mu}$ at random from the training sample, and compute

$$h_i = \sum_{j=1}^{n} w_{ij} x_j \quad \text{for} \quad \forall i = 1, ..., k \tag{63}$$

3. **The Winner Takes All Procedure**

   * Modify only the weight w_{mj} which recieves

$$h_m = \max_i h_i \tag{64}$$

   as

$$w_{mj}^{\text{new}} = w_{mj}^{\text{old}} + \delta w_{mj} \tag{65}$$

   where

$$\delta w_{mj} = \epsilon (x_k - w_{ik}) \tag{66}$$

   and normalize them again.

4. Repeat 2 and 3 until a predetermined number of steps is reached.

Note that

- The difference between this algorithm and perceptron learning is that the input set cannot be classified a priori in a positive or a negative set or in any of several different clusters.
- When we monitor the number of updates for each weight vector in order to make the process as balanced as possible, we call this learning with conscience.

- A numerical excersise:

  Assume three input neurons and three output neurons with w_{ij}

$$\begin{pmatrix} 0.23 & -0.72 & -0.51 \\ -0.49 & 0.33 & -0.16 \\ 0.23 & 0.68 & 0.84 \end{pmatrix}. \tag{67}$$

  Then under an input $(-1, 1, -1)$,

  ∗ (1) obtain h_m;

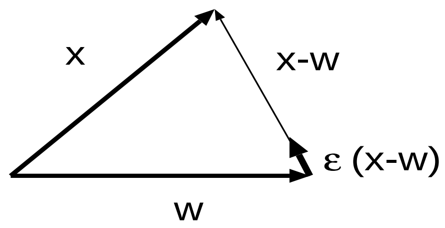
  ∗ (2) modify weights.

- How it works?

  – When we take $\mathbf{x}$ and $\mathbf{w}$ as

$$\begin{aligned} \mathbf{x} &= (x_1, x_2, \cdots, x_{N_i},) \\ \mathbf{w} &= (w_{i1}, w_{i2}, \cdots, w_{iN_i},) \end{aligned}$$

  the equation (66) implies a vector such as: Therefore $\mathbf{w} + \delta\mathbf{w}$ makes $\mathbf{w}$



  become closer to $\mathbf{x}$, and

  – the more $\mathbf{w}$ is closer to $\mathbf{x}$, the larger the inner product h_i becomes.

  – What will happen when multiple $\mathbf{x}^\mu$ is repeatedly given?

(2) Kohonen's Self-organizing Network:

Simulation of topology preserving maps in the brain.

- Kohonen Learning Algorithm

  The n-dimensional weight vectors $w_1, .., w_m$ of the m computing neurons are selected at random. An initial radius r, a learning constant η, and a neighborhood funciton ϕ are selected.

  1. Select an input vector ζ using the desired probability distribution over the input space.

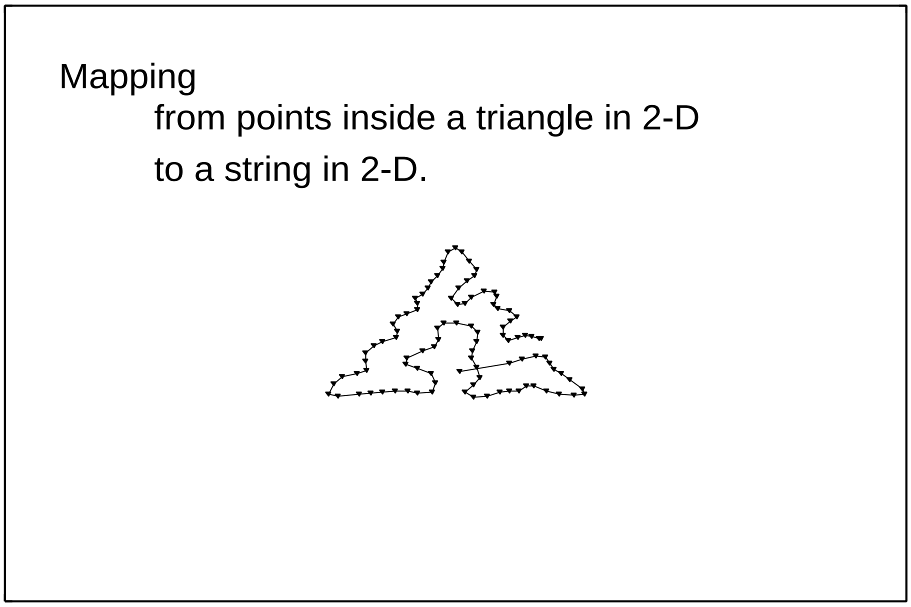
  2. The neuron k with the maximum excitation is selected, that is, for which the distabnce between w_i and ζ is minimal for $i = 1, \cdots, m$.

  3. The weight vectors are updated using the nighborhood function and the update rule

  $$w_i = w_i + \eta\phi(i,k)(\zeta - w_i)\text{for}i = 1, \cdots, m \tag{68}$$

  4. Stop if the maximum number of iterations has been reached, otherwise modify η and ϕ as sceduled and continue with step 1.

- An example:



(3) Unsupervised Reinforcement Learning

- Assume that a set of empirical data is given which consists of the n-dimensional vectors $(x_1, \cdots, x_m)$.
- *The first principal component* of this set of vectors is a vector $\mathbf{w}$ which maximizes

$$\frac{1}{m}\sum_{i=1}^{m}\|w_i x_i\|^2 \tag{69}$$

- The second principal component is computed by subtracting from each vector x_i its projection on the first principal component.

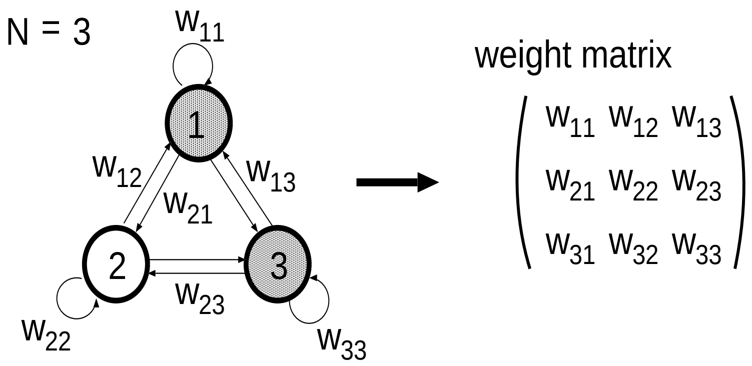
- architecture
  - n input neurons which are given $x_1, \cdots, x_n$
  - one ouput neuron which receives

$$\sum_{i=1}^{n} w_i x_i \tag{70}$$

- Algorithm — Computation of the first principal component
  - Let $\mathbf{x}$ be a n-dimensional vector.
  - The vector $\mathbf{w}$ is initialized randomly ($w_i \neq 0$).
  - A learning constant γ with $0 < \gamma \leq 1$ is selected.
  - A vector x is selected randomly from x
  - The scalar product $\phi = xw$ is computed the new weight vector is $w + \gamma\phi(x - \phi w)$
  - go to update making γ smaller.

□ Fully-connected Neural Networks.

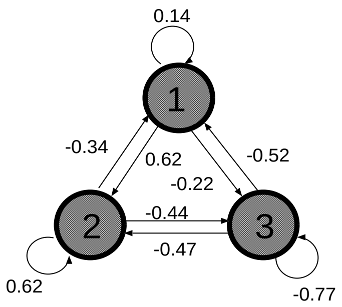
- Toy Model of Three Neurons:

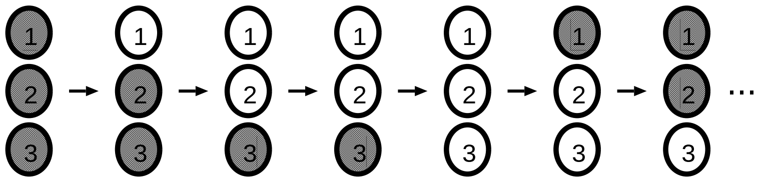


- State Transition

$$s_i(t+1) = sgn\left(\sum_{j \neq i}^{N} w_{ij} \cdot s_j(t)\right).$$

- Example 1

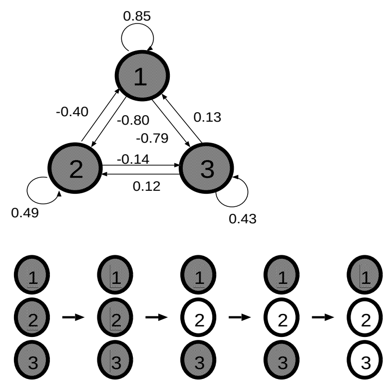




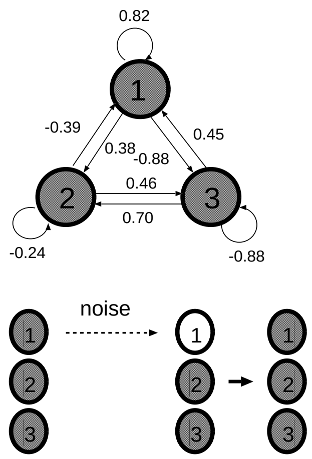
An example of state transition in the above toy example:

$$\begin{aligned} s_1(1) &= sgn\ (\ w_{11} \cdot s_1(0) + w_{12} \cdot s_2(0) + w_{13} \cdot s_3(0)\ ) \\ &= sgn\ (\ 0.14 \cdot 1 + (-0.34) \cdot 1 + (-0.52) \cdot 1\ ) \\ &= -1 \end{aligned}$$

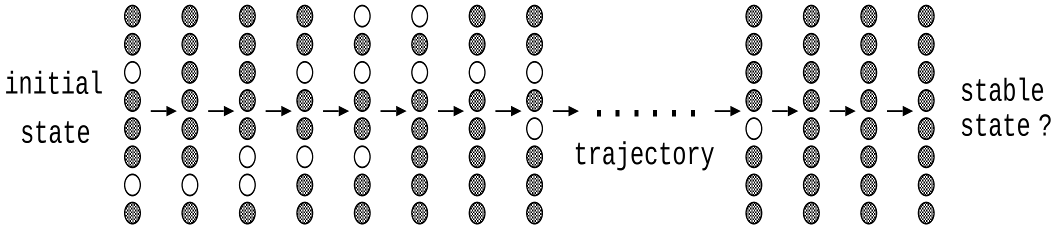
- Example 2

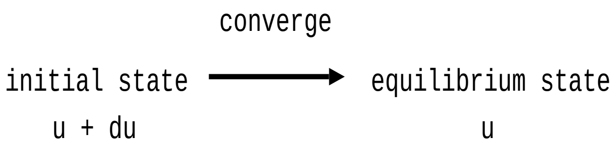


- Example 3

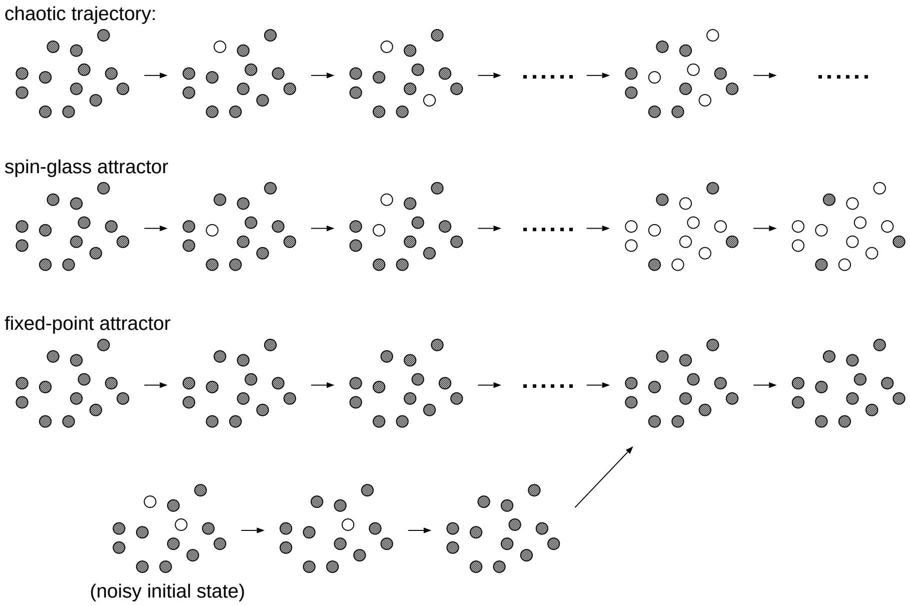


□ Associative Memory as a Dynamical System





□ A category of trajectories



□ What is Associative Memory

- N neurons that are fully connected store N-bit of p out of 2^N patterns simultanously as *fixed point attractors.*[7]
- Convergence to the initial state of the noisy stored pattern implies an *error correcting capability.*
- How can we determine weights.
  - Hebbian Weights

    $$w_{ij} = \frac{1}{N} \sum_{\nu=1}^{p} \xi_i^\nu \xi_j^\nu \ \ (i \neq j), \ \ w_{ii} = 0.$$

    In the above toy example:

    $$\begin{pmatrix} 0 & 0.11 & 0.11 \\ 0.11 & 0 & 0.11 \\ 0.11 & 0.11 & 0 \end{pmatrix}$$

    An interpretation of Hebb's rule:
    - A strong weight affects neurons at both ends to be same states.
  - Many other solutions of a *weight configuration* to store a same set of patterns exist.
    - Each can *recall* from
      - partial information;
      - incorrect information
    - but has a different
      - *basin of attraction*;
      - *storage capacity.*

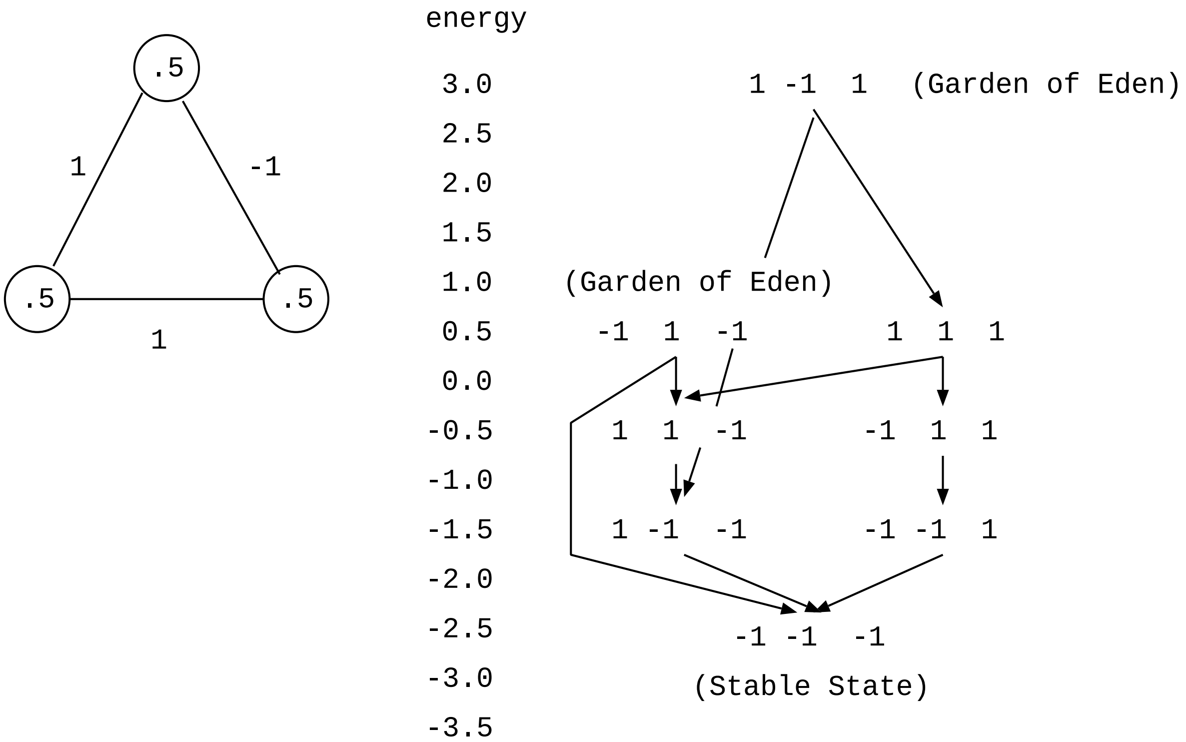
[7]Here we restrict patterns to bipolar patterns.

□ (Virtual) Energy of Network State

Assuming fully connected N neurons, given all weight values w_{ij} from i-th to j-th neuron and threshold of i-th neuron θ_i, each state $\{x_i\}$ correspond to an energy level E as:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_ix_j - \sum_{j=1}^{N}\theta_i x_i \tag{71}$$

For example:



Excersise 7 *Obtain state of the Garden-of-Eden and stable state(s) under three neurons with* $w_{12} = w_{21} = w_{23} = w_{32} = -1$*,* $w_{13} = w_{31} = 1$ *and with all threshold beeing* -0.5*.*

□ Solving Combinatorial Problem

The fully connected neural network model so far we've studied can also solve combinatorial problems.

e.g.

- **Traveler's Salesman Problem (TSP)**[8]
  - as an NP-complete[9] combinatorial problem
    - ∗ A salesman should visit every city once but only once such that his traveling route is the shortest[10].
  - Representation of a rout by a $N \times N$ matrix
    - ⋆ The rout 1-3-6-4-5-2 of 6-city TSP is represented as

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{73}$$

      where "ij-element is one" means that the i-th city is in the j-th location in the tour.

  - Implementation by a fully connected neural network with N neurons.
    - ∗ $S_{ij}(t)$ state of neuron corresponds to the ij-element of the matrix.
    - ∗ Starting with a random configuration of the network state, it is expected to converge to the matrix corresponding to the shortest route.

[8] We can apply the problem to "how we connect each device in LSI in which many, say 10^5, divices are included."

[9] Nondeterministic polynomial ,that is, the effort to solve the problem grows exponentially with size instead of a power of the size (polynomial).

[10] With N cities, the number of all the possible route is

$$(N-1)! = \sqrt{2\pi N} \exp(N \ln N - N) \tag{72}$$

i.e., *not deterministic in polynomial time.*

Then the energy of our neural network is

$$E = -\frac{1}{2} \sum_{i,k}^{i \neq k} \sum_{\alpha,\beta}^{\alpha \neq \beta} w_{i\alpha,k\beta} S_{i\alpha} S_{k\beta} \tag{74}$$

This can be expanded with appropriate A, B, and C as

$$\begin{aligned} E = & - \frac{1}{2} \sum_{i,k,\alpha}^{i \neq k} d_{ik} S_{i\alpha} (S_{k(\alpha-1)} + S_{k(\alpha+1)}) \\ & - \frac{A}{2} \sum_{i,\alpha,\beta}^{\alpha \neq \beta} S_{i\alpha} S_{i\beta} - \frac{B}{2} \sum_{i,k,\alpha,}^{i \neq k} S_{i\alpha} S_{k\alpha} \\ & - \frac{C}{2} \sum_{i,\alpha} (S_{i\alpha} S_N)^2 \end{aligned} \tag{75}$$

$$\tag{76}$$

Then

- $\star$ 1st term means total length of the tour;
- $\star$ 2nd term vanishes iff the raw includes only one “1”;
- $\star$ 3rd term vanishes iff the column includes only one “1”;
- $\star$ 4th term takes its minimum when exactly N neurons are active.

- Simpler Version of the TSP

(1) Multiflop problem:

  - Only one out of n neurons is on with all the others off at a moment.
  - When all weight values are -2,
    * a unit that is set to 1 inhibits the other units; or
    * if the network is started with all neurons set to zero, then the excitation of every neuron is zero, which makes the first neuron to be asynchronously selected flip its state to 1 and no other neuron can change its state after this first neuron has been set to 1

(2) The eight rooks problem:

  - A generalization of multiflop;
  - If the connection of each neuron to other neurons in the same row or column have the weight -2, all others have a weight zero and all neurons have the threshold -1, then
    * Any neuron set to 1 inhibits any other neurons in the same row or column; or
    * if a row or column is all set to 0 when one of its neuron is selected it will immediately switch its state to 1