Artificial Neural Networks

References

Machine learning by Tom Mitchell: Chapter 4

Russell & Norvig: 20.5

(slides Chapter 19 old edition book)

Elements of Artificial Neural Networks by K. Mehrotra, C.K. Mohan & S. Ranka

Various online resources

Goals of Neural Networks

- ❒ The main goal is to realize an artificial intelligent system using the human brain as the model.

- ❒ There are three basic problems in this area:

  - ❍ What kind of structure or model should we use ?

  - ❍ How to train or design the neural networks ?

  - ❍ How to use neural networks for information acquisition ?

Thus Fundamental Problems for a Given Neural Model

- How to represent information?

- How to characterize the computational capability of the model?

- How to achieve learning in the model?

Where can neural networks help?

- where we can't formulate an algorithmic solution.
- where we can get lots of examples of the behaviour we require.
- where we need to pick out the structure from existing data.

Roots of work on NNs are in:

- **Neurobiological studies**:
  - How do nerves behave when stimulated by different magnitudes of electric current?
  - Is there a minimal threshold needed for nerves to be activated?
  - How do different nerve cells communicate among each other?

- **Psychological studies**:
  - How do animals learn, forget, recognize and perform various types of tasks?

- **Psycho-physical:**experiments help to understand how individual neurons and groups of neurons work.

- **McCulloch and Pitts** introduced the first mathematical model of single neuron, widely applied in subsequent work. (we' ll look at this)

Neural nets can be used to answer the following:

- Pattern recognition: Does that image contain a face?
- Classification problems: Is this cell defective?
- Prediction: Given these symptoms, the patient has disease X
- Forecasting: predicting behavior of stock market
- Handwriting: is character recognized?
- Optimization: Find the shortest path for the TSP.

Why Neural Networks? (I)

- Fast system with slow elements
  - Massively parallel processing
  - Decentralized computing
- Reliable system with non-reliable elements
  - Robust to noises or disturbances
  - Fault tolerant
  - Make good (reasonable, rational) decisions based on incomplete and ambiguous information

Why Neural Networks? (II)

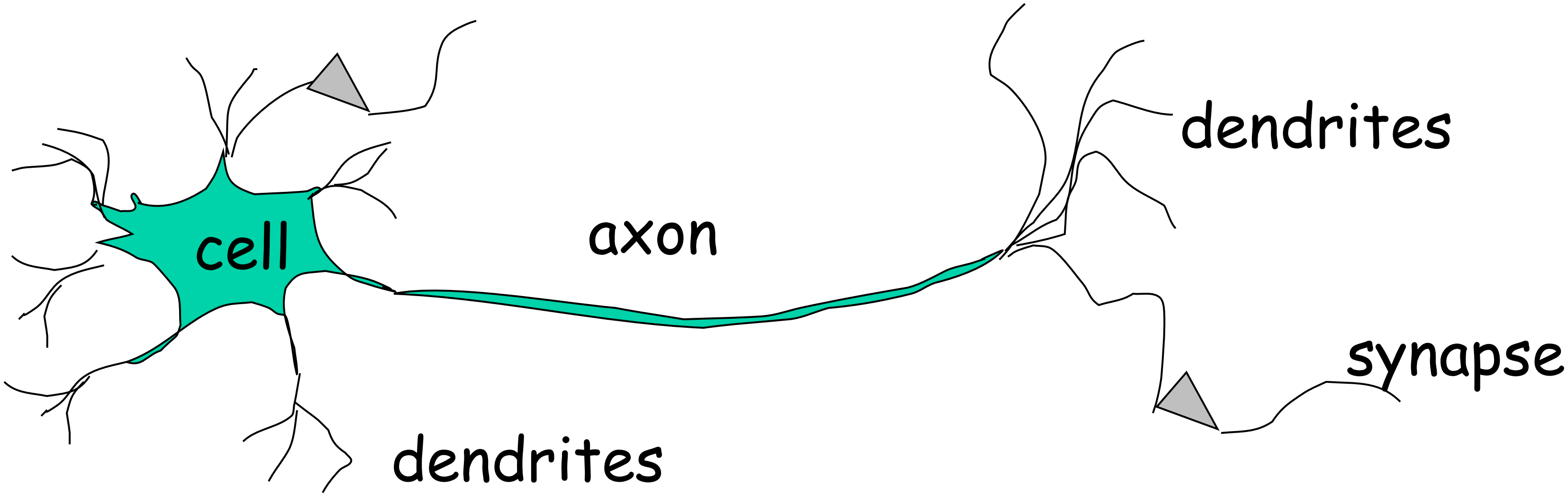
- Intelligent system with non-intelligent elements
  - Each element is a simple unit
  - The whole is an intelligent system
- NN is a different kind of computer analog to human brain
- NN can learn from experience, and can solve different kinds of problems after learning
- NN can learn in real-time, and can adapt to changing environments flexibly

Consider biological neurons

human information processing system consists of biological brain neuron: basic building block – cell that communicates infor. to and from various parts of body

- Neuron switching time ~ .001 second
- Number of neurons ~ 10^10
- Connections per neuron ~ 10 ^4-5
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough
  - much parallel computation

Biological neurons



Biological Neurons

- human information processing system consists of brain neuron: basic building block
  - cell that communicates information to and from various parts of body
- Simplest model of a neuron: considered as a threshold unit –a processing element (PE)
- Collects inputs & produces output if the sum of the input exceeds an internal threshold value

Artificial Neural Nets (ANNs)

- Many neuron-like PEs units
  - Input & output units receive and broadcast signals to the environment, respectively
  - Internal units called hidden units since they are not in contact with external environment
  - units connected by weighted links (synapses)
- A parallel computation system because
  - Signals travel independently on weighted channels & units can update their state in parallel
  - However, most NNs can be simulated in serial computers

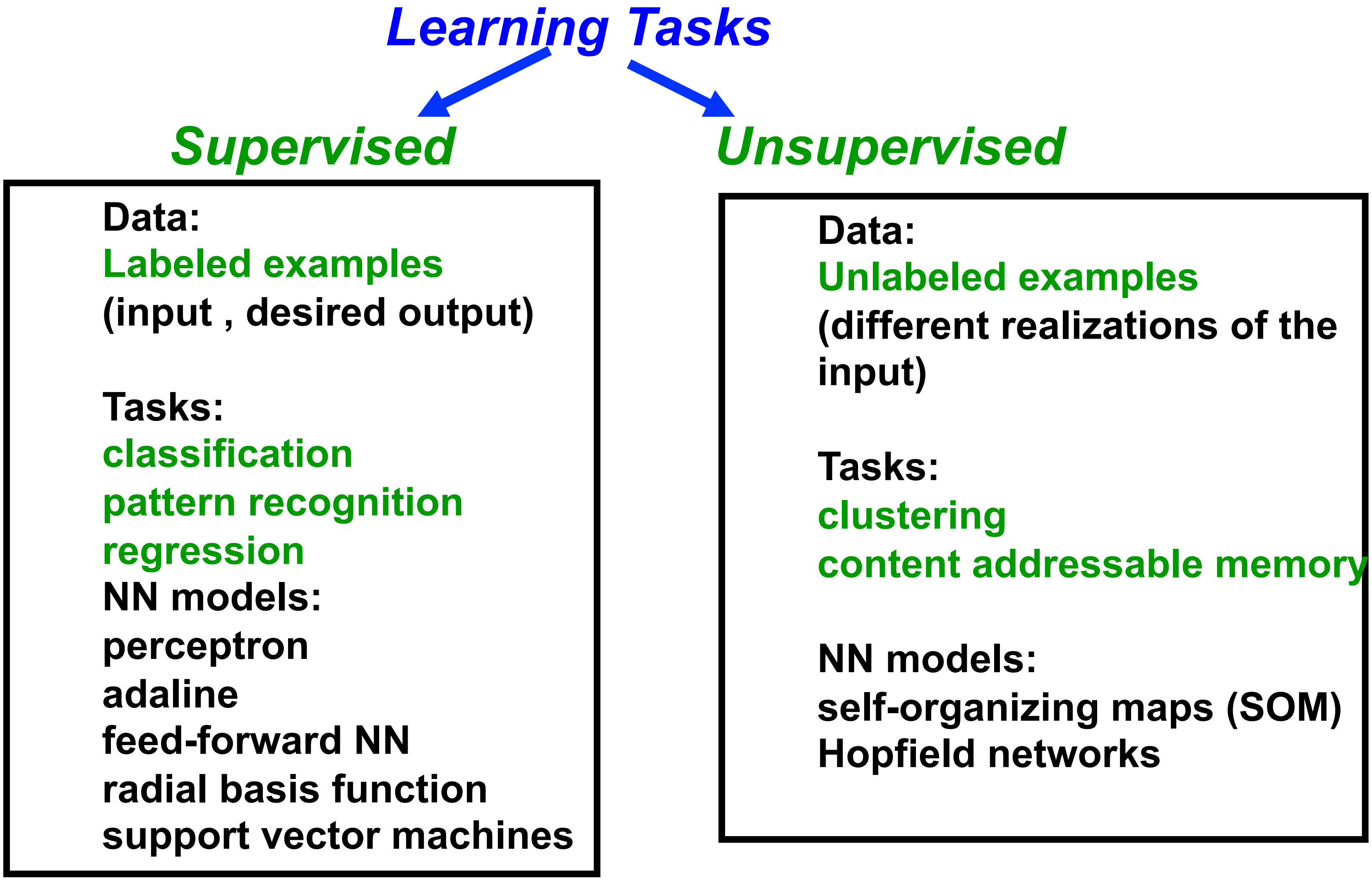
Properties of ANNs

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically
- Input is a high-dimensional discrete or real-valued (e.g, sensor input)

Properties of ANNs II

- Output is discrete or real-valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown

Learning in Neural Nets



Learning Tasks

Supervised

Data:
Labeled examples
(input , desired output)

Tasks:
classification
pattern recognition
regression
NN models:
perceptron
adaline
feed-forward NN
radial basis function
support vector machines

Unsupervised

Data:
Unlabeled examples
(different realizations of the input)

Tasks:
clustering
content addressable memory

NN models:
self-organizing maps (SOM)
Hopfield networks

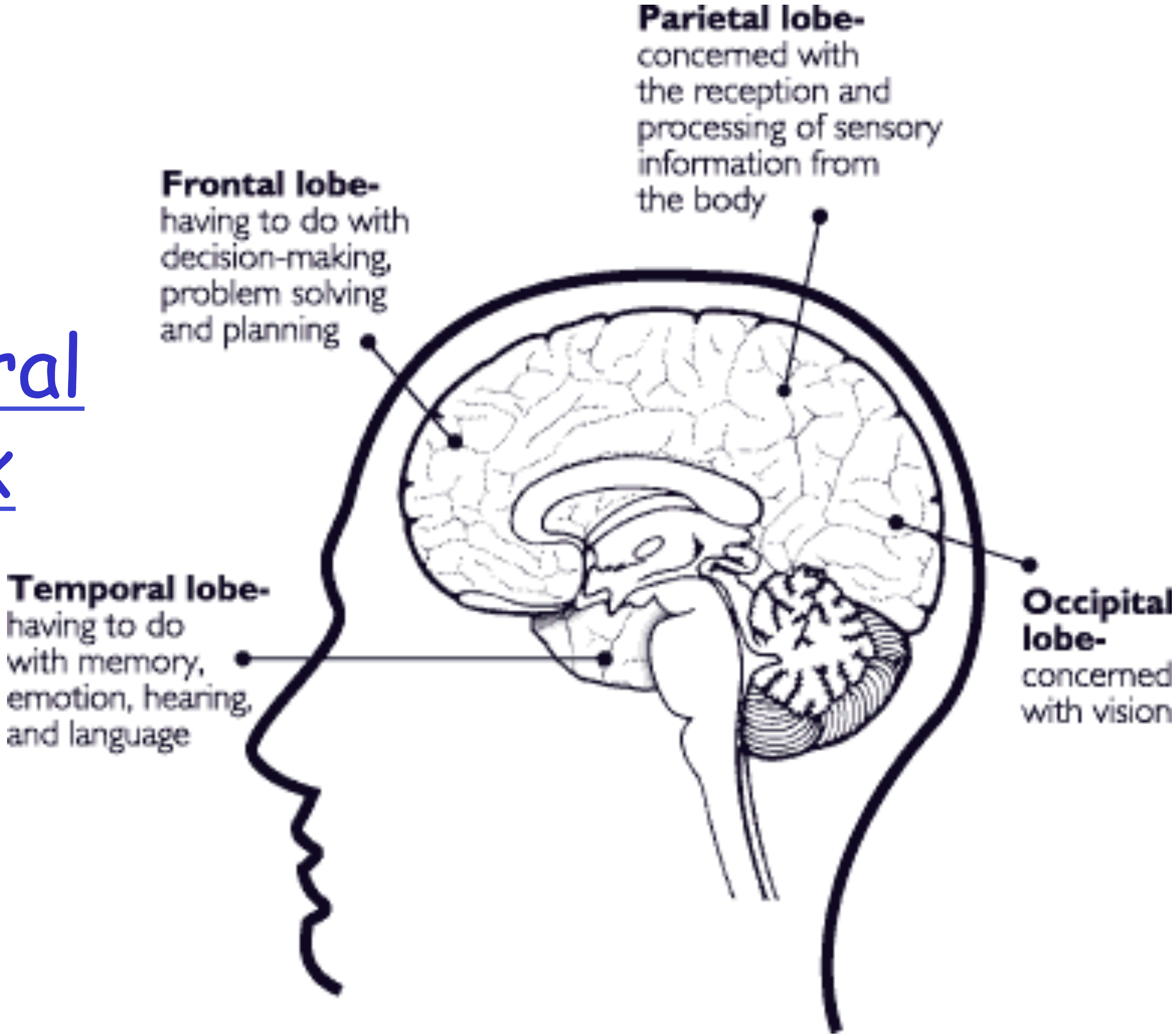
This class:

- Feed-forward neural networks (main focus)
- Self-organizing maps (COSC 4P80 & quick overview in COSC 4P76)
- Evolving neural networks (COSC4P80 & some project topic in COSC 4P76)
- Recurrent neural networks

Self-Organizing Maps

- It’s developed by observing how neurons work in the brain and in ANNs:
  - The firing of neurons impact the firing of other neurons that are near it
  - Neurons that are far apart seem to inhibit each other
  - Neurons seem to have specific non-overlapping tasks

Cerebral Cortex



Parietal lobe- concerned with the reception and processing of sensory information from the body

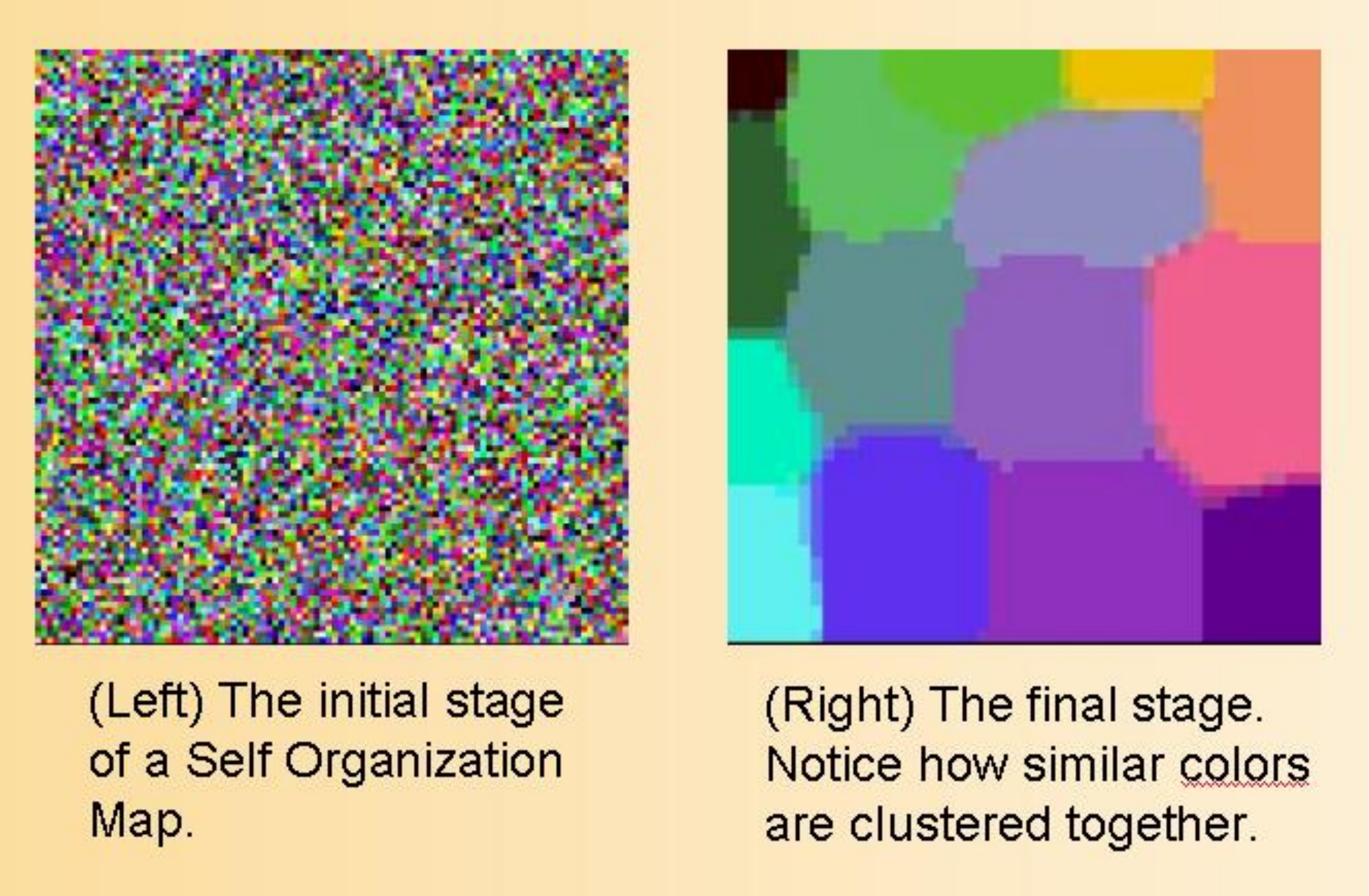
Frontal lobe- having to do with decision-making, problem solving and planning

Temporal lobe- having to do with memory, emotion, hearing, and language

Occipital lobe- concerned with vision

http://www.health.org/pubs/qdocs/mom/TG/intro.htm

Example:Result



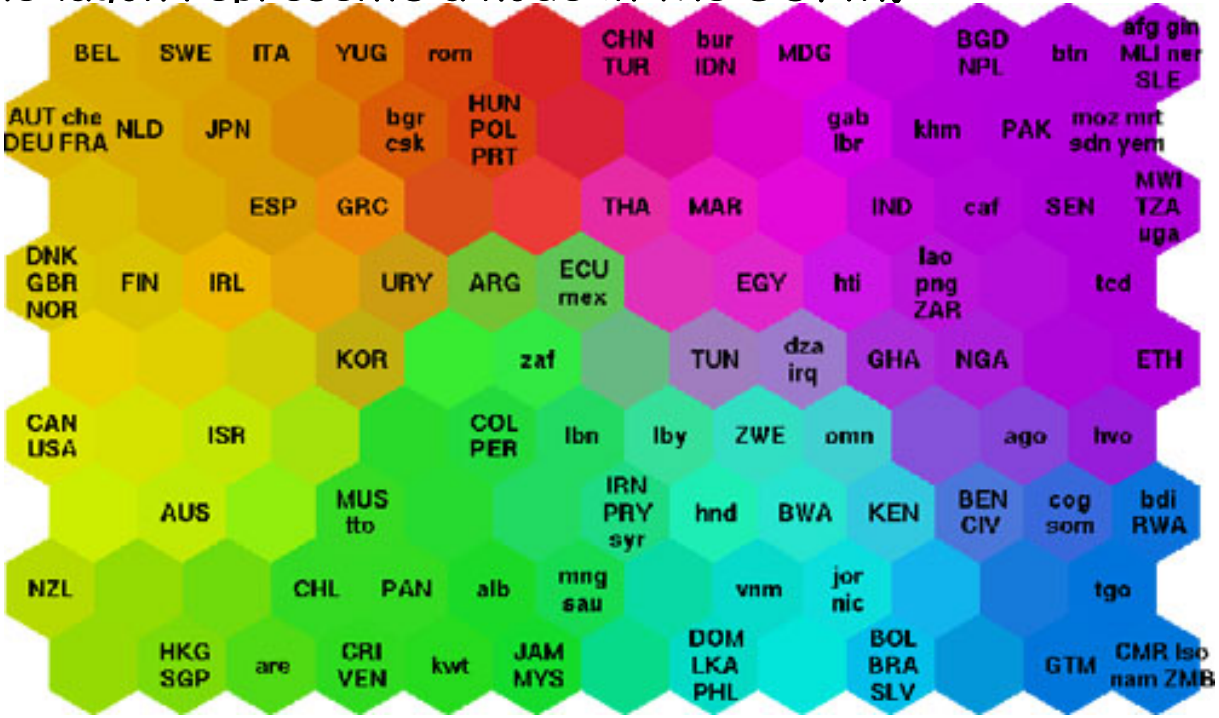
(Left) The initial stage of a Self Organization Map.

(Right) The final stage. Notice how similar colors are clustered together.

Applications of SOM (1)

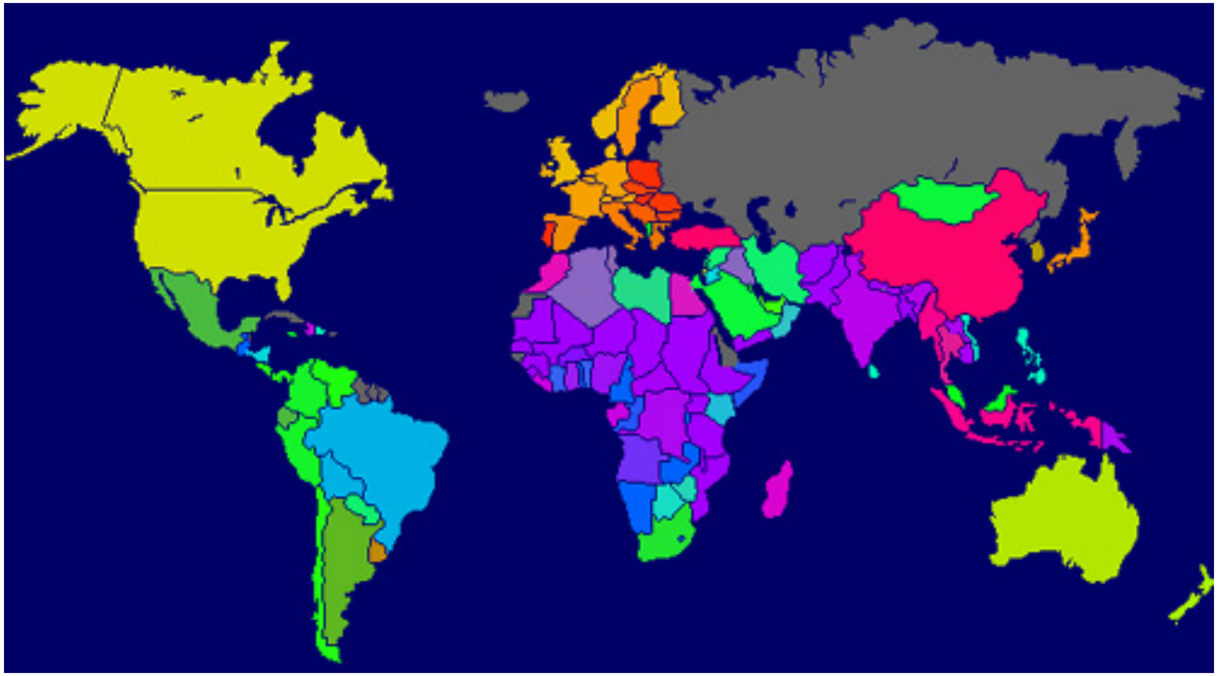
- A SOM has been used to classify statistical data describing various quality-of-life factors such as state of health, nutrition, educational services etc

- Each hexagon represents a node in the SOFM.



Applications of SOFM (1b)

- The colour information can then be plotted onto a map of the world:

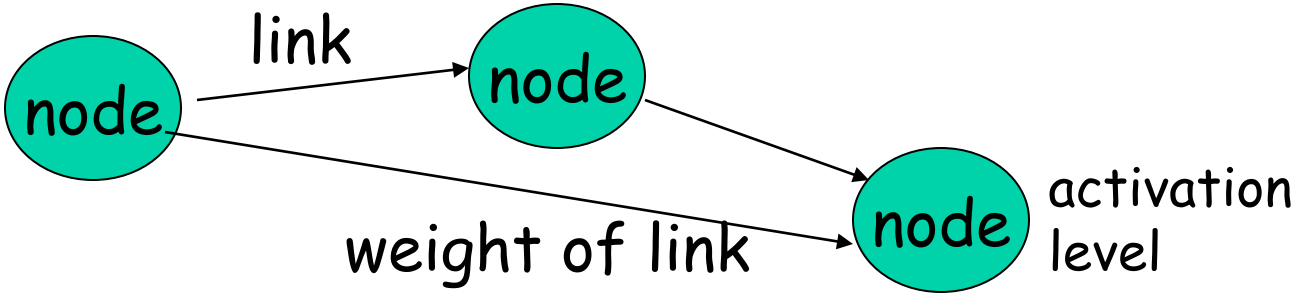


Feed-forward neural networks

- ❒ ANN representations
- ❒ Perceptron Training
- ❒ Multilayer networks and Backpropagation algorithm
- ❒ Backpropagation algorithm: Tricks & variants
- ❒ ANN Applications

Neuron

node = unit



node

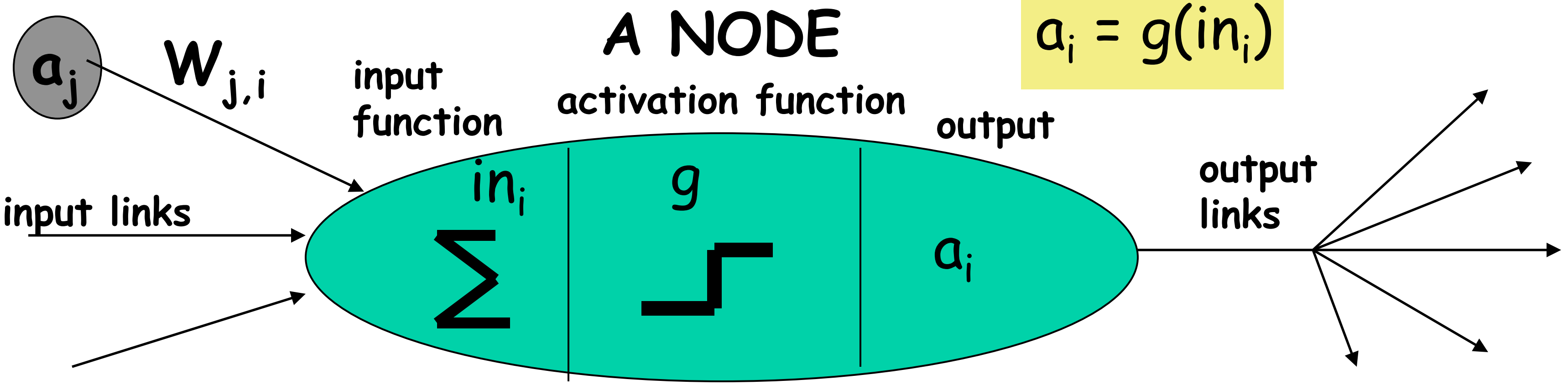
link

node

node

activation level

weight of link



A NODE

$a_i = g(in_i)$

a_j

$W_{j,i}$

input function

activation function

output

input links

in_i

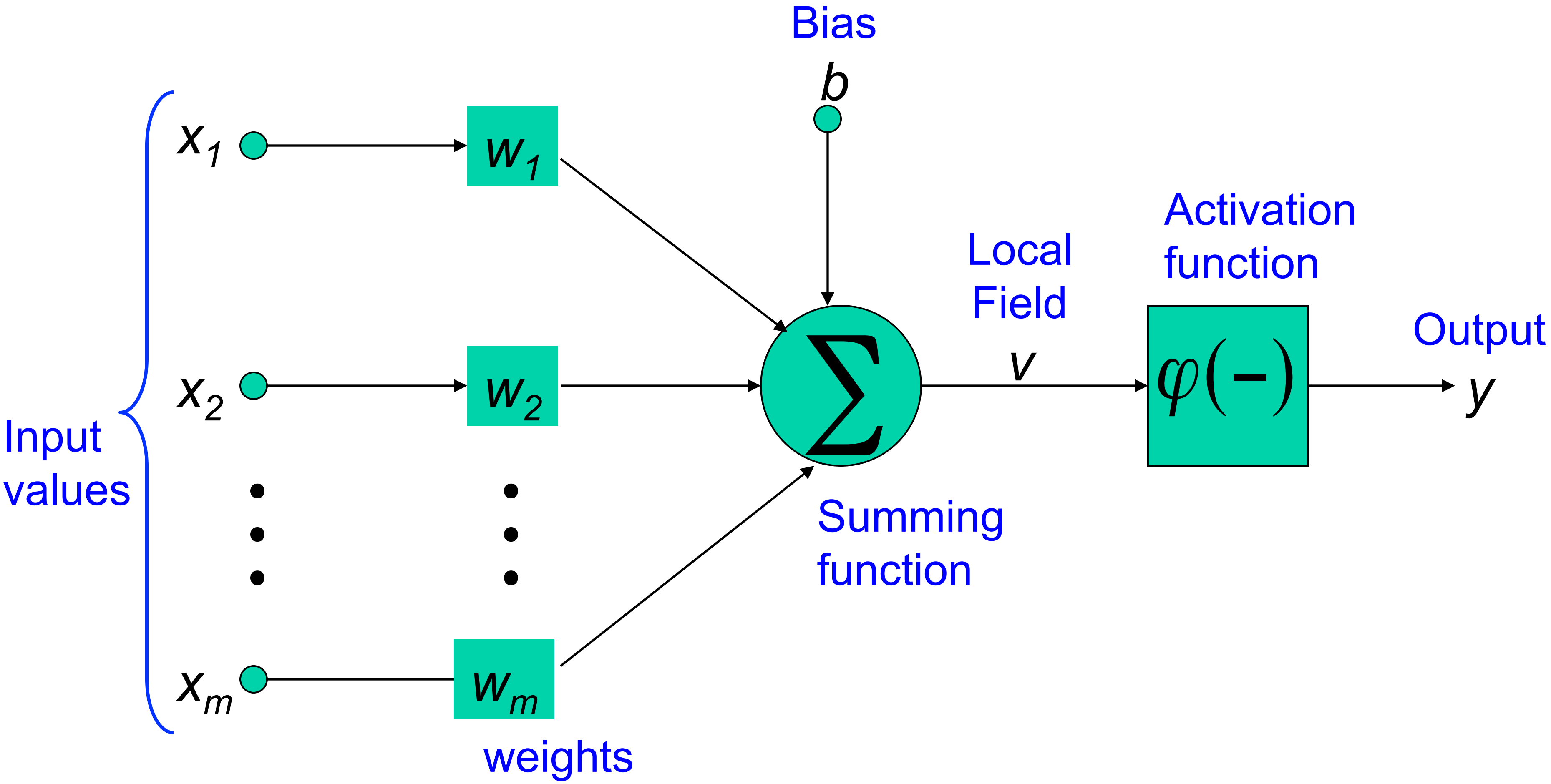
Σ

g

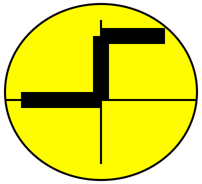
a_i

output links

Neuron


Bias
b
x_1
w_1
Local
Field
v
Activation
function
$\varphi(-)$
Output
y
x_2
w_2
$\sum$
Input
values
Summing
function
x_m
w_m
weights

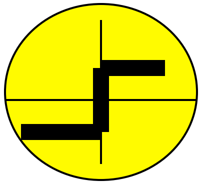
g = Activation functions for units



Step function

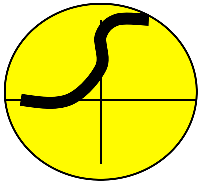
(Linear Threshold Unit)

step(x) = 1, if x >= threshold
0, if x < threshold



Sign function

sign(x) = +1, if x >= 0
-1, if x < 0



Sigmoid function

sigmoid(x) = 1/(1+e^{-x})

Network architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent

- The architecture of a neural network is linked with the learning algorithm used to train

Note:

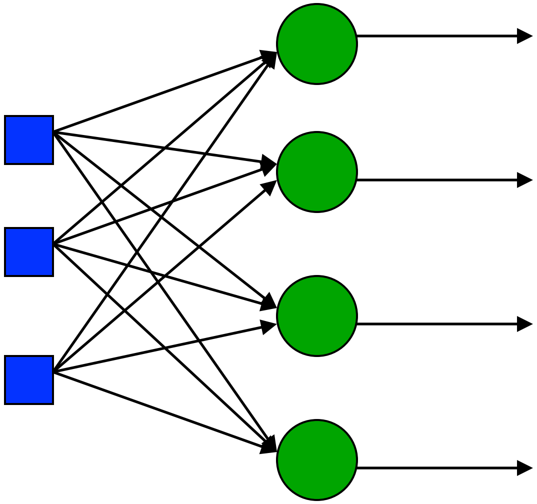
recurrent: links form arbitrary topologies e.g., Hopfield Networks and Boltzmann machines

Recurrent networks: can be unstable, or oscillate, or exhibit chaotic behavior e.g., given some input values, can take a long time to compute stable output and learning is made more difficult…. However, can implement more complex agent designs and can model systems with state

We will focus more on feed- forward networks

Single Layer Feed-forward

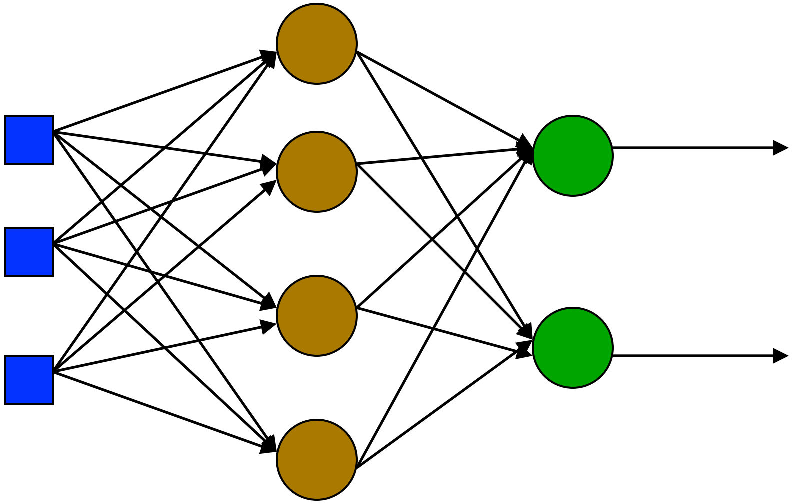
Input layer of source nodes



Output layer of neurons

Multi layer feed-forward

3-4-2 Network



Input layer

Output layer

Hidden Layer

Feed-forward networks:

Advantage: lack of cycles = > computation proceeds uniformly from input units to output units.

-activation from the previous time step plays no part in computation, as it is not fed back to an earlier unit

- simply computes a function of the input values that depends on the weight settings –it has no internal state other than the weights themselves.

- fixed structure and fixed activation function g: thus the functions representable by a feed-forward network are restricted to have a certain parameterized structure

Neural Network Learning

- Objective of neural network learning: given a set of examples, find parameter settings that minimize the error.
  - The aim is to obtain a NN that generalizes well, that is, that behaves correctly on new instances of the learning task.

- **Programmer specifies**

  - numbers of units in each layer
  - connectivity between units,

- **Unknowns**

  - connection weights

Therefore A NN is specified by:

- an architecture: a set of neurons and links connecting neurons. Each link has a weight,
- a neuron model: the information processing unit of the NN,
- a learning algorithm: used for training the NN by modifying the weights in order to solve the particular learning task correctly on the set of training examples.

Learning Algorithms

Depend on the network architecture:

❒ Error correcting learning (perceptron)
❒ Delta rule (AdaLine, Backprop)
❒ Competitive Learning (Self Organizing Maps)

Perceptron

Rosenblatt (1958) defined a perceptron to be a machine that learns, using examples, to assign input vectors (samples) to different classes, using linear functions of the inputs

Minsky and Papert (1969) instead describe perceptron as a stochastic gradient-descent algorithm that attempts to linearly separate a set of n-dimensional training data.

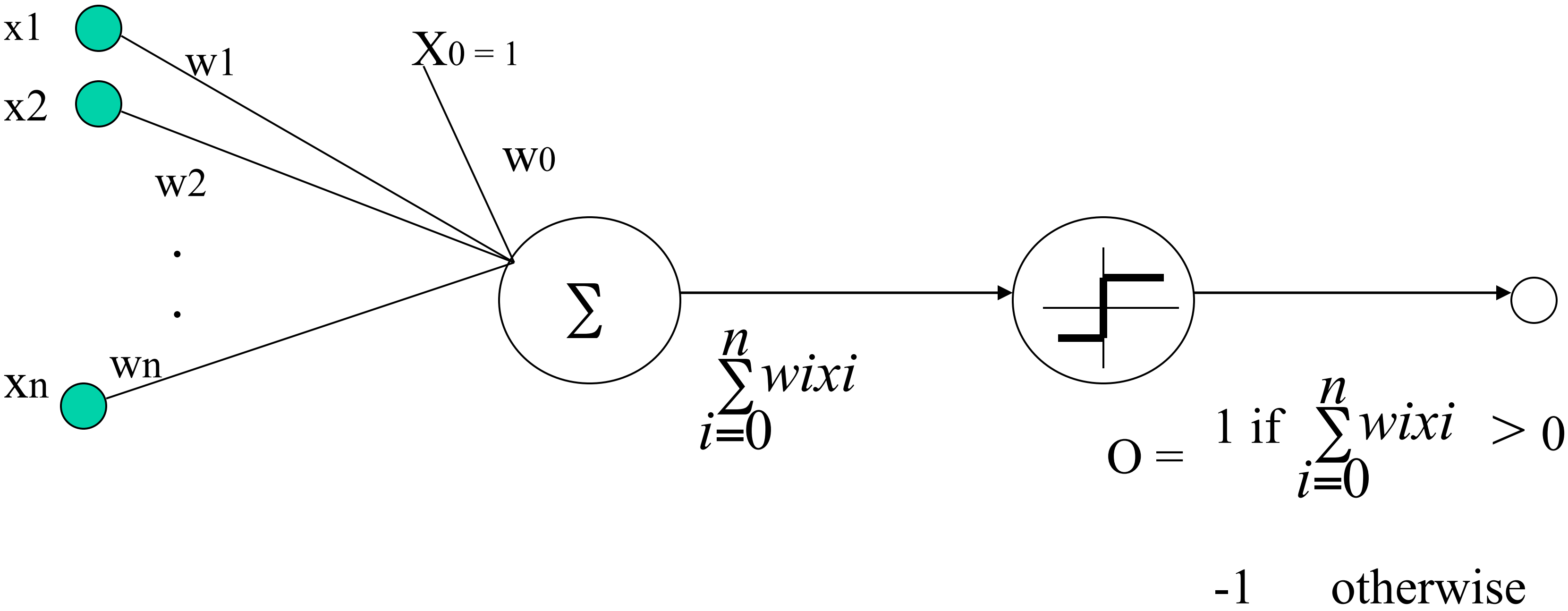
Perceptrons

- **Perceptrons** are single-layer feedforward networks
- Each output unit is independent of the others
- Can assume a single output unit
- Activation of the output unit is calculated by:

- $O = \text{Step}_0\left(\sum_{j} W_j \, x_j\right)$

where x_j is the activation of input unit j, and we assume an additional weight and input to represent the threshold

Perceptron



$$O = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} wixi > 0 \\ -1 \quad \text{otherwise} \end{cases}$$

Figure 4.2 (from Mitchell) A perceptron

Linear Separable

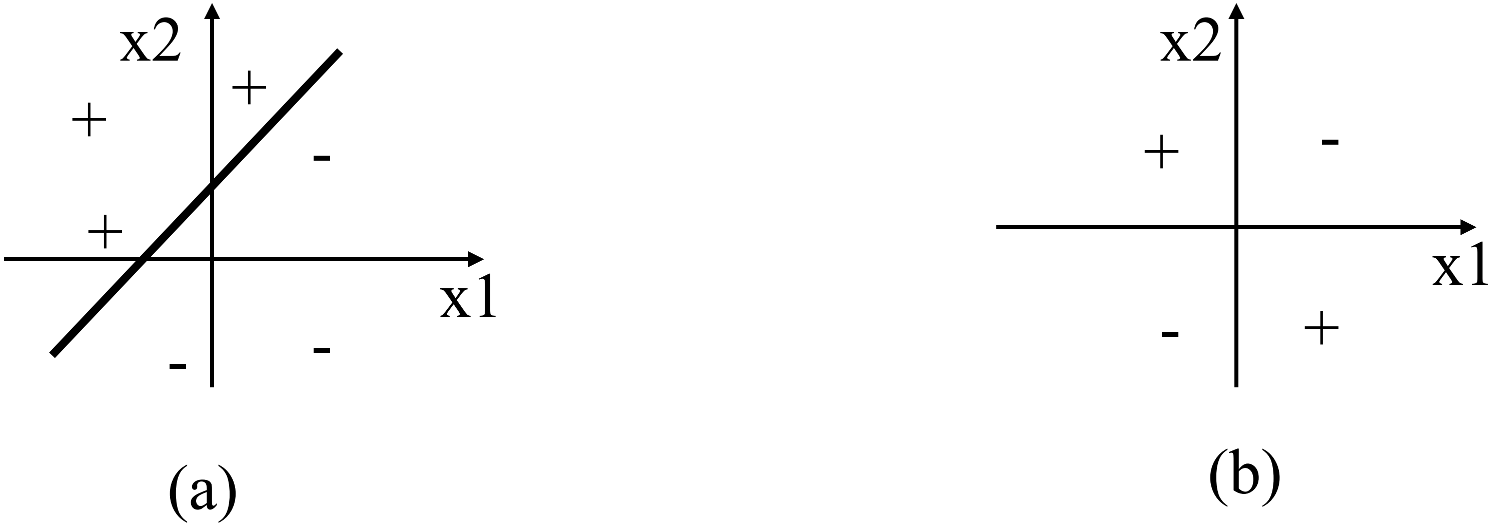


Figure 4.3 (Mitchell)

some functions not representable

- e.g., (b) not linearly separable

How can perceptrons be designed?

- The Perceptron Learning Theorem (Rosenblatt, 1960): Given enough training examples, there is an algorithm that will learn any linearly separable function.

Theorem 1 (Minsky and Papert, 1969) The perceptron rule converges to weights that correctly classify all training examples provided the given data set represents a function that is linearly separable

Learning in Perceptrons

Algorithm:

```
    1.randomly assign weights to initial network (usually values range[-0.5,0.5])
    2.repeat until all examples correctly predicated or stopping criterion is met
        for each example e in training set do
          i).O = neural-net-output(network, e)
          ii).T = observed output values from e
          iii).update-weights in network based on e, O, T
```

Note: Each pass through all of the training examples is called one epoch

Learning in Perceptrons

- Inputs: training set $\{(x_1,x_2,\ldots,x_n,t)\}$
- Method
  - Randomly initialize weights w(i), -0.5<=i<=0.5
  - Repeat for several epochs until convergence:
    - for each example
      - Calculate network output o.
      - Adjust weights:

$$\left.\begin{aligned}\Delta w_i &= \overset{\text{learning rate}}{\eta}\,\overset{\text{error}}{(t-o)}\,x_i \\ w_i &\leftarrow w_i + \Delta w_i\end{aligned}\right\}\ \text{Perceptron training rule}$$

Perceptrons

- Perceptron training rule guaranteed to succeed if
  - Training examples are linearly separable
  - Sufficiently small learning rate

Multi-layer, feed-forward networks

- Perceptrons are rather weak as computing models since they can only learn linearly-separable functions.

- Thus, we now focus on multi-layer, feed forward networks of non-linear sigmoid units: i.e.,

$$g(x) = \frac{1}{1+e^{-x}}$$

Multi-layer feed-forward networks

Multi-layer, feed forward networks extend perceptrons i.e., 1-layer networks into n-layer by:

- **Partition units into layers 0 to L such that;**

- **lowermost layer number, layer 0 indicates the input units**

- **topmost layer numbered L contains the output units.**

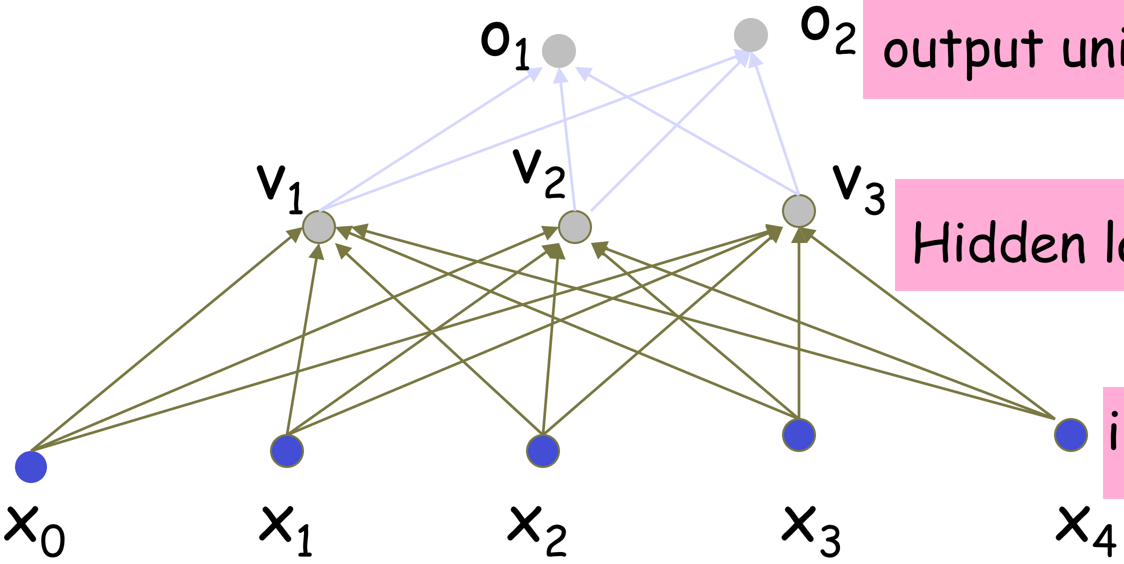
- **layers numbered 1 to L are the hidden layers**

- **Connectivity means bottom-up connections only, with no cycles, hence the name"feed-forward" nets**

- **Input layers transmit input values to hidden layer nodes hence do not perform any computation.**

Note: layer number indicates the distance of a node from the input nodes

Multilayer feed forward network



output units

Hidden layer

input units

Hidden Units

- Hidden units are nodes that are situated between the input nodes and the output nodes.

- Given too many hidden units, a neural net will simply memorize the input patterns.

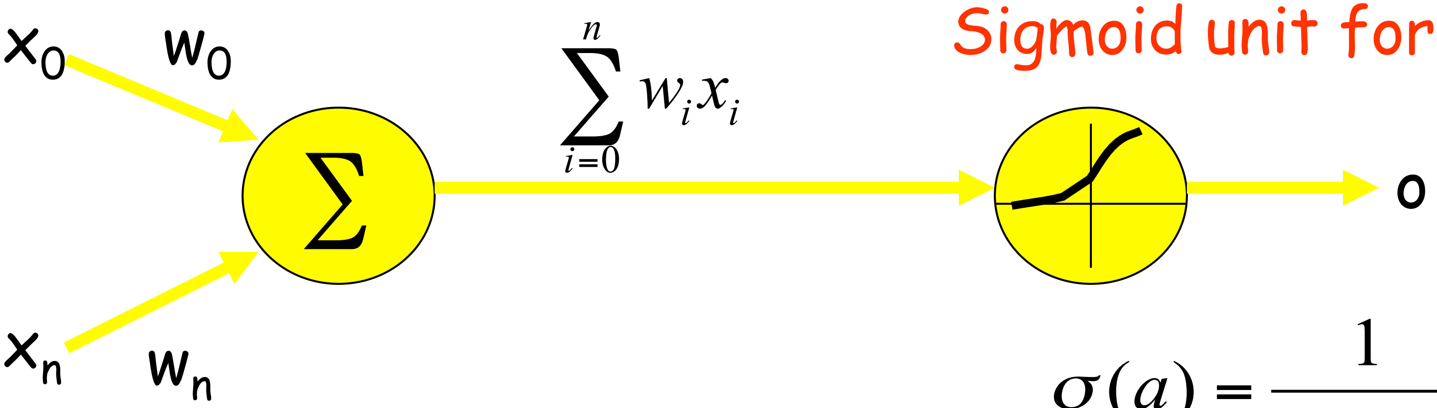
- Given too few hidden units, the network may not be able to represent all the necessary representations.

Multi-layer feed-forward networks

- Multi-layer feed-forward networks can be trained by **back-propagation** provided the activation function g is a differentiable function.
  - Threshold units don't qualify, but the sigmoid function does.

- **Back-propagation learning** is a **gradient descent search** through the parameter space to minimize the sum-of-squares error.
  - Most common algorithm for learning algorithms in multilayer networks

Sigmoid units



x_0

w_0

x_n

w_n

Σ

$$\sum_{i=0}^{n} w_i x_i$$

o

Sigmoid unit for g

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

$$\frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$$

This is g' (the basis for gradient descent)

Back-propagation Learning

- Inputs:
  - Network topology: includes all units & their connections
  - Some termination criteria
  - Learning Rate (constant of proportionality of gradient descent search)
  - Initial parameter values
  - A set of classified training data
- Output: Updated parameter values

Learning in backprop

- Learning in backprop is similar to learning with perceptrons, i.e.,
  - Example inputs are fed to the network.
    - If the network computes an output vector that matches the target, nothing is done.
    - If there is a difference between output and target (i.e., an error), then the weights are adjusted to reduce this error.
    - The key is to assess the blame for the error and divide it among the contributing weights.
- The error term (T - o) is known for the units in the output layer. To adjust the weights between the hidden and the output layer, the gradient descent rule can be applied as done for perceptrons.
- To adjust weights between the input and hidden layer some way of estimating the errors made by the hidden units in needed.

Learning in Back-propagation

```
1.Initialize the weights in the network (often randomly)
2.repeat
    for each example e in the training set do
      i.O = neural-net-output(network, e) ; forward pass
      ii.T = teacher output for e
      iii.Calculate error (T - O) at the output units
      iv.Compute $wj = wj + \alpha * Err * Ij$ for all weights from
         hidden layer to output layer;backward pass
      v.Compute $wj = wj + \alpha * Err * Ij$ for all weights from input layer
        to hidden layer; backward pass continued
      vi.Update the weights in the network
    end
3.until all examples classified correctly or stopping criterion met
4.return(network)
```

Estimating Error (see separate example)

- Main idea: each hidden node contributes for some fraction of the error in each of the output nodes.
  - This fraction equals the strength of the connection (weight) between the hidden node and the output node.

$$\text{error at hidden node j} = \sum_{i \in outputs} w_{ij} \delta_i$$

where δ_i is the error at output node i.

Number of training pairs needed?

Difficult question. Depends on the problem, the training examples, and network architecture. However, a good rule of thumb is:

$$\frac{w}{p} = e$$

Where W = No. of weights; P = No. of training pairs, e = error rate

For example, for e = 0.1, a net with 80 weights will require 800 training patterns to be assured of getting 90% of the test patterns correct (assuming it got 95% of the training examples correct).

How long should a net be trained?

- The objective is to establish a balance between correct responses for the training patterns and correct responses for new patterns. (a balance between memorization and generalization).

- If you train the net for too long, then you run the risk of overfitting.

- In general, the network is trained until it reaches an acceptable error rate (e.g., 95%)

Implementing Backprop – Design Decisions

1. Choice of *r*
2. Network architecture
   a) How many Hidden layers? how many hidden units per a layer?
   b) How should the units be connected? (e.g., Fully, Partial, using domain knowledge
3. Stopping criterion – when should training stop?

Backpropagation

- Performs 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)

- Minimizes error over training examples
  - Will it generalize well to subsequent examples
    - Guarding against overfitting needed

- Training can take thousands of iterations (epocs) → Slow!

- Using network after training is very fast

Convergence of Backpropagation

Gradient descent to some local minimum

- Perhaps not global minimum...
- Add momentum
- Stochastic gradient descent
- Train Multiple Nets with different initial weights

Back-propagation Using Gradient Descent

- Advantages
  - Relatively simple implementation
  - Standard method and generally works well
- Disadvantages
  - Slow and inefficient
  - Can get stuck in local minima resulting in sub-optimal solutions

Learning rate

- Ideally, each **weight** should have its own learning rate (*extra notes on tricks for BP)*

- As a substitute, each neuron or each layer could have its own rate

Determining optimal network structure

Weak point of fixed structure networks: poor choice can lead to poor performance

Too small network: model incapable of representing the desired Function

Too big a network: will be able to memorize all examples but forming a large lookup table, but will not generalize well to inputs that have not been seen before.

Thus finding a good network structure is another example of a search problems.
Some approaches to search for a solution for this problem include Genetic algorithms
But using GAs can be very cpu-intensive.

- **Search: hardest task is to obtain a suitable representation for search space in terms of nodes and weights in a network**

e.g if a NN is to be used for game playing,
Inputs: describe the current state of the board game

desired output pattern: identifies the best possible move to be made.

weights in the network can be trained based on an evaluation of the quality of previous moves made by the network in response to various input patterns

Setting the parameter values

- How are the weights initialized?
- Do weights change after the presentation of each pattern or only after all patterns of the training set have been presented?
- How is the value of the learning rate chosen?
- When should training stop?
- How many hidden layers and how many nodes in each hidden layer should be chosen to build a feedforward network for a given problem?
- How many patterns should there be in a training set?
- How does one know that the network has learnt something useful?

When should neural nets be used for learning a problem

- If instances are given as attribute-value pairs.
  - Pre-processing required: Continuous input values to be scaled in [0-1] range, and discrete values need to converted to Boolean features.

- Noise in training examples.

- If long training time is acceptable.

Neural Networks: Advantages

- Distributed representations
- Simple computations
- Robust with respect to noisy data
- Robust with respect to node failure
- Empirically shown to work well for many problem domains
- Parallel processing

Neural Networks: Disadvantages

- Training is slow

- Interpretability is hard

- Network topology layouts ad hoc

- Can be hard to debug

- May converge to a local, not global, minimum of error

- Not known how to model higher-level cognitive mechanisms

- May be hard to describe a problem in terms of features with numerical values