Neural Nets: Learning Goals

1. Definition and Network Structure
2. Single-layer Feed-forward networks
3. Multilayer Feed-forward networks

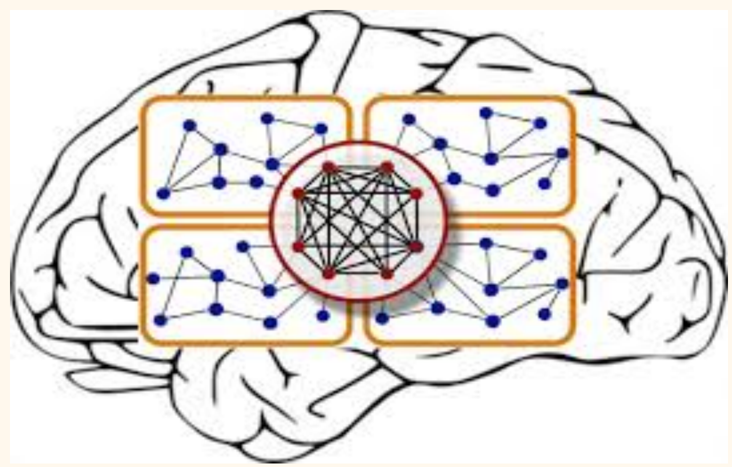
- Russell 18.7

Many Classification Approaches

- Decision trees
- Linear models, e.g., regression, ridge, lasso
- Neural Nets
- Naïve Bayes
- kNN
- SVM
- Ensembles
- Random forests
- Hidden Markov Models
- Conditional Random Fields

Simulating Human Brains?

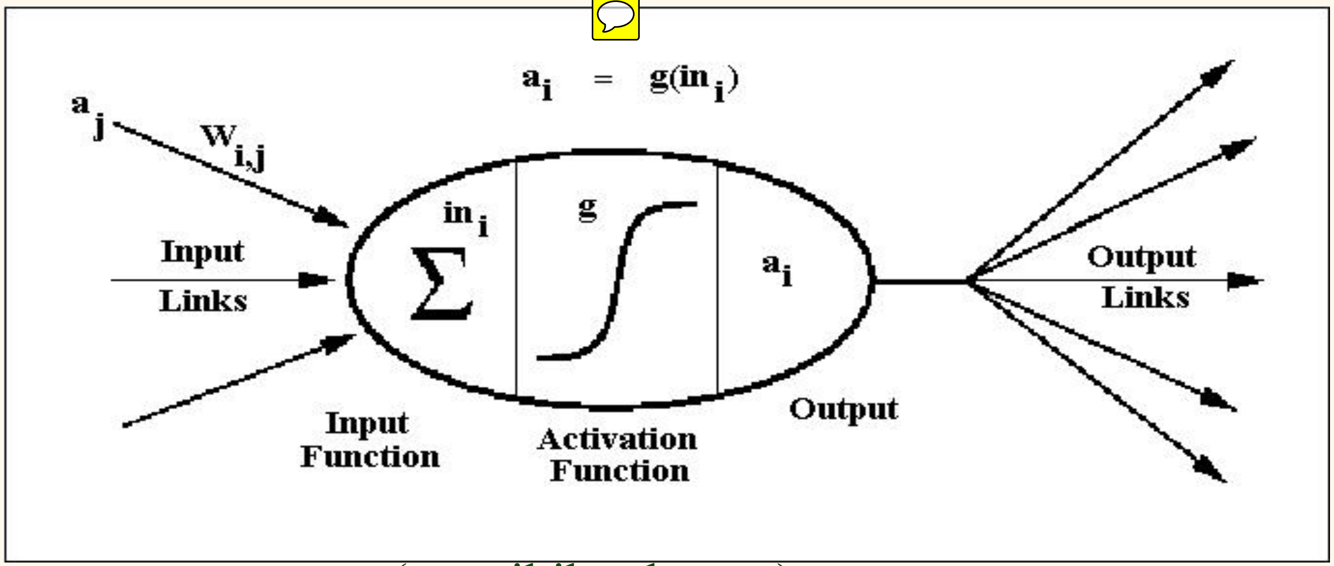
- To build a classifier, why not do what a human brain does?
- In particular, neuroscience reveals the existence of massively connected networks of neurons
- In 1943, McCulloch and Pitts proposed a simple mathematical model for neurons, now known as neural networks



(journal.frontiersin.org)

Structure of a Network Unit

- A Neural network composed on nodes/units connected by links
- A link from unit i to unit j propagates an activation a_i to possibly trigger an activation a_j *(caution: i and j swapped below; but use Fig 18.19)*



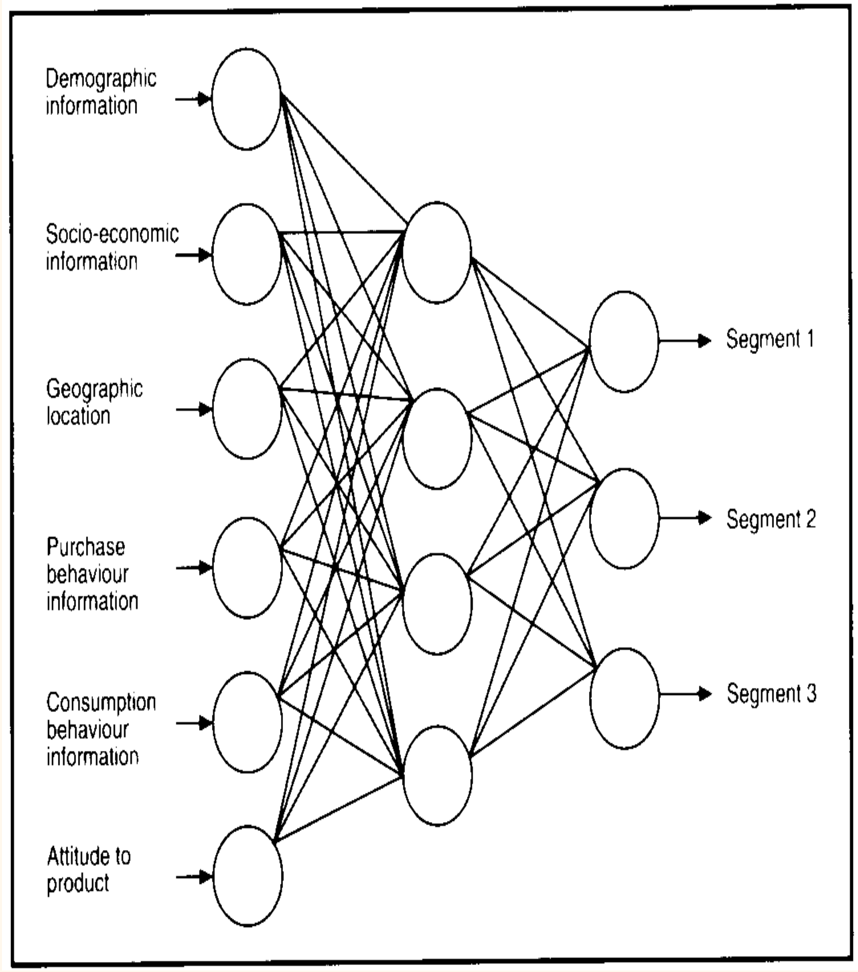
(en.wikibooks.org)

Input and Output Activations

- Each input link a_i has a weight $w_{i,j}$ determining the strength and sign of the connection
- Unit j computes a weighted sum of its inputs:
  $input_j = \Sigma^n_{i=0} (w_{i,j}\, a_i)$
- Unit j then applies an activation function g to the weighted sum of inputs to derive its output:
  $a_j = g(input_j) = g(\Sigma^n_{i=0} (w_{i,j}\, a_i))$
- g can be a hard threshold (Fig 18.17), in which case the unit is called a perceptron, or a logistic function, in which case a sigmoid perceptron

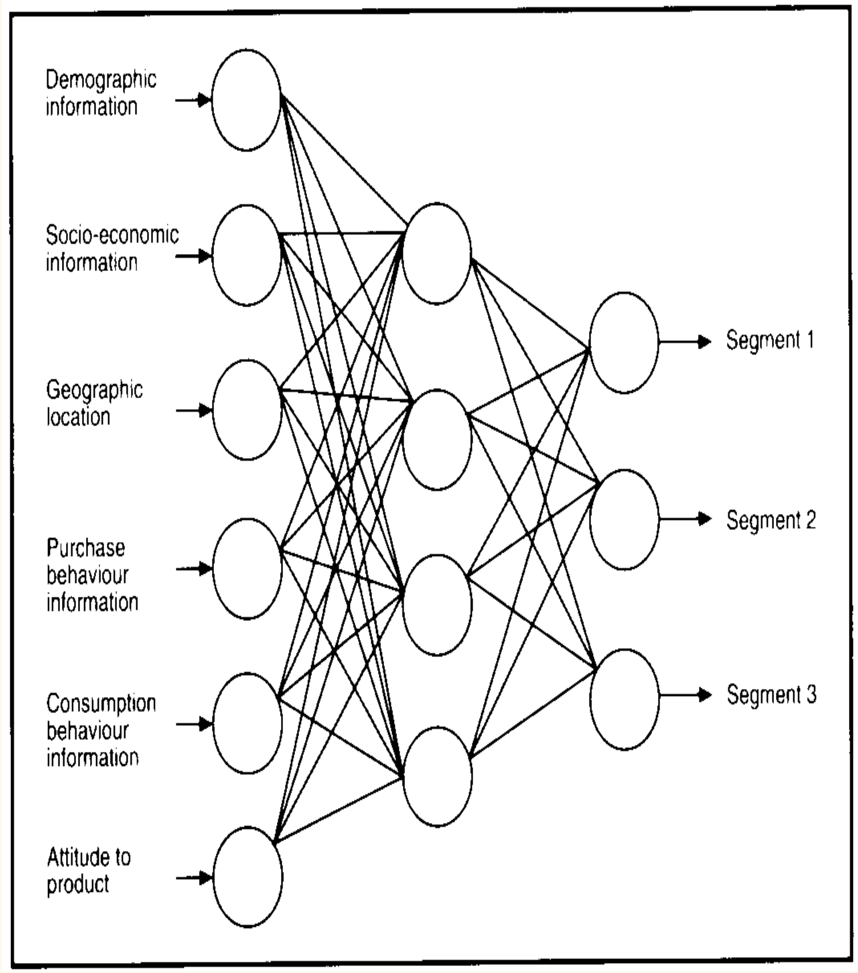
Feed-forward and Recurrent Networks

- This network determines to which segment a customer belongs
- This is an example of a feed-forward network
  - No loops, forming an acyclic graph
- A recurrent network has loops and may take time to reach a steady state



Layers in Feed-forward Networks

- Our example has a hidden layer of units
- Common to have units arranged in layers, i.e., inputs from the preceding layer and outputs to the next layer
- Why do we need hidden layers?



Hidden Layers

- Layers add complexity to what we can model
- Suppose we have 1 hidden layer in between the inputs and outputs
- call the input vector x, the hidden layer activations h, and the output activation y
- have some function f that maps from x to h and another function g that maps from h to y
- So the hidden layer's activation is $f(x)$ and the output of the network is $g(f(x))$
- $g(f(x))$ can make computation that g() and f() can't do individually – will return to this issue later

E.g., Single-layer Boolean Function Networks

- Gain insights on the simplest networks – boolean logical operators, i.e., AND, OR, NOT
- Consider an adder unit

x	y	*Carry*	*Sum*
0	*0*	*0*	*0*
0	*1*	*0*	*1*
1	*0*	*0*	*1*
1	*1*	*1*	*1*

- Thus, Carry(x,y) = AND(x,y) and Sum(x,y) = XOR(x,y)

Linearly Separability

- Easy for a 1-layer NN to learn AND
- Observe that AND is linearly separable
- So are OR and NOT

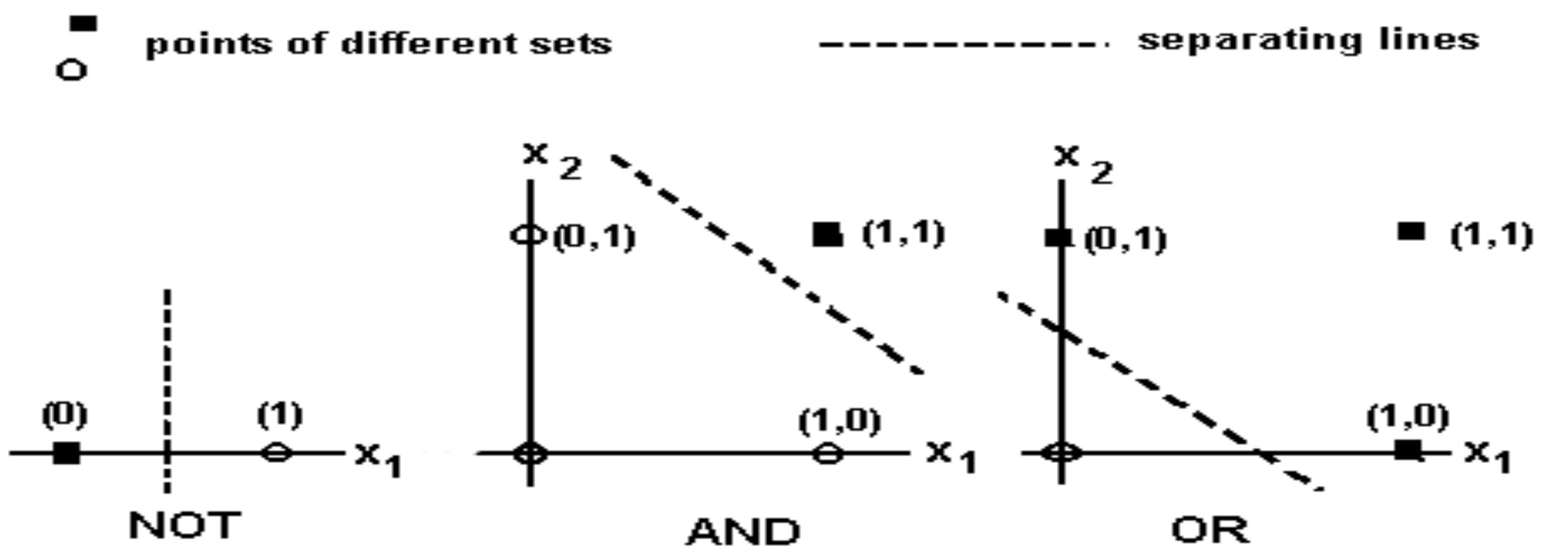


Fig 6. Pattern spaces for gates

(www.hpcc.org)

Linear Separability (2)

- XOR is not captured in a single-layer NN because it is not linearly separable
- But adding hidden layers can! How?
- XOR(x,y) is true when:
  - First layer functions:
    - (i) OR(x,y)
    - (ii) NOT(AND(x,y))
  - Second layer function: both (i) and (ii) are true with AND

Expressiveness of NNs

- XOR shows that hidden layers can help
- But in general if training data not linearly separable, NNs can take effort to learn, as compared with decision trees (Fig 18.22)
- The reverse also true, i.e., NNs can learn other scenarios faster than decision trees
- E.g., a majority function (Fig 18.22)
  - Output = 1 iff $n/2$ inputs are 1
  - $w_i = 1$ for each of the n inputs; and $w_0 = -n/2$

Expressiveness of NNs (2)

- Our discussion on logical operators show the “ideal” weights for an NN to capture the operator
- Fig 18.22 shows the speed for an NN to learn from the training data and to converge to the “ideal” weights
- Given NNs are good for linear separable scenarios, NNs are very good linear regression
- Multi-layer NNs are also very good for non-linear regression (Fig 18.23)

Learning in NNs

- For single-layer NNs, the perceptron learning rule sufficient for learning the weights
- For multi-layer NNs, the learning much harder (sec 18.7.4, not required)
- Expressiveness comes from hidden layers, which require more complex back-propagation
- Learning the structure of an NN also not trivial
- Cross-validation often used to select the best structure (sec 18.7.5, not required)

Concluding Remarks: Big Data

- NNs are often used for scenarios that are known to be non-linear
- Optimized for large NNs
- One weakness of NNs is that the hidden layers may not have natural meanings, unlike the paths in a decision tree
- So while NNs may be good for classification performance, they may not be as good in generating insights for users