Connectionist Language Processing

Lecture 6: **Pattern Association Networks**

Matthew W. Crocker
crocker@coli.uni-sb.de
Harm Brouwer
brouwer@coli.uni-sb.de

Overview

Pattern Associators:

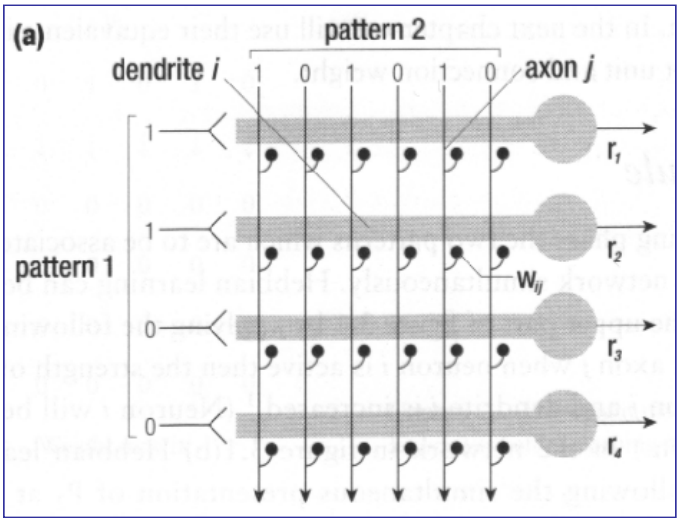
- 2-layer networks
- Networks as matrices
- Associating distributed representations
- Hebbian learning
- Generalisation in learning
- Biological plausibility

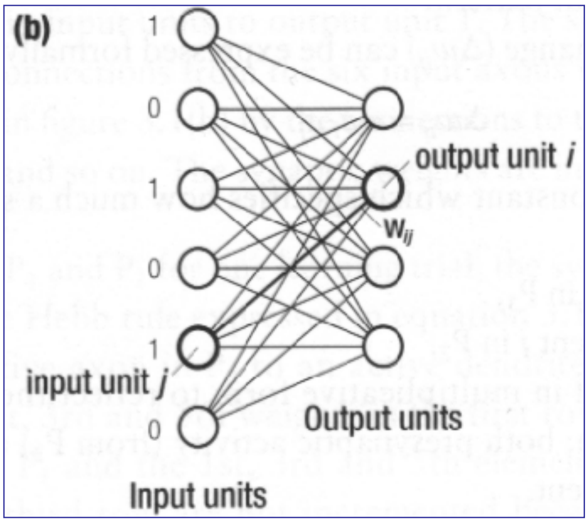
Competitive networks and unsupervised learning

Pattern Associators

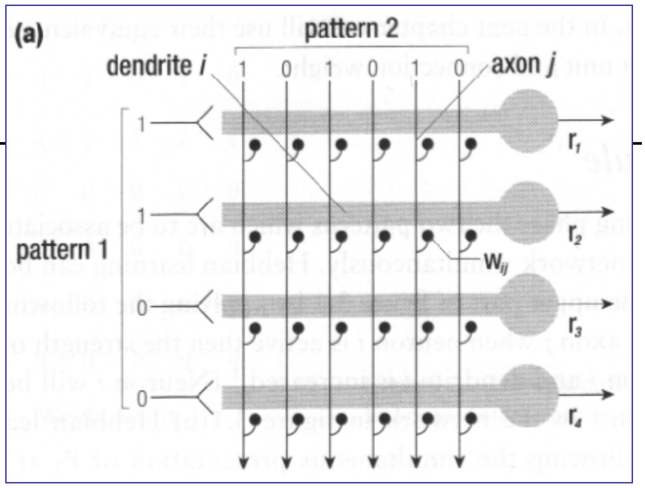
Learn to associate one stimulus with another, e.g.:

- Sight of chocolate associates with taste of chocolate
- The string “yacht” associates with the pronunciation /y/ /o/ /t/
- …





Hebb’s rule



- The idea behind Hebbian learning is simple: **reinforcement**
- The two patterns to be associated are presented simultaneously
- If there is activity on input axon ***j***, when neuron ***i*** is active, then the connection weight w_{ij} (between axon ***j*** and dendrite ***i***)is increased
- The Hebb rule:
  - a_i is the activity of element *i* in P_1
  - a_j is the activity of element *j* in P_2
  - ε is the learning rate parameter

$$\Delta w_{ij} = \varepsilon a_i a_j$$

An example

- Assume binary neuron activations (0 or 1)
- Suppose the sight of chocolate is represented as: (1 0 1 0 1 0)
- The taste of chocolate is represented as (1 1 0 0)
- We can represent the weights as a 6x4 matrix of "synapses"

$$\Delta w_{ij} = \varepsilon a_i a_j$$

Weights before learning:

		1	0	1	0	1	0
		↓	↓	↓	↓	↓	↓
1	→	0	0	0	0	0	0
1	→	0	0	0	0	0	0
0	→	0	0	0	0	0	0
0	→	0	0	0	0	0	0

Weights after learning:

		1	0	1	0	1	0
		↓	↓	↓	↓	↓	↓
1	→	1	0	1	0	1	0
1	→	1	0	1	0	1	0
0	→	0	0	0	0	0	0
0	→	0	0	0	0	0	0

- Assume that ε=1

Recall from a Trained Matrix

- $\text{Netinput}_i = \sum_j a_j w_{ij}$ .... but this is just the ***dot product*** of 2 vectors, i.e.:

$$(1\ 0\ 1\ 0\ 1\ 0) \bullet (1\ 0\ 1\ 0\ 1\ 0)$$
$$= (1\times1 + 0\times0 + 1\times1 + 0\times0 + 1\times1 + 0\times0) = 3$$

- Thus for the recall cue (1 0 1 0 1 0), the output pattern is:

1	0	1	0	1	0	
↓	↓	↓	↓	↓	↓	
1	0	1	0	1	0	→
1	0	1	0	1	0	→
0	0	0	0	0	0	→
0	0	0	0	0	0	→

- If we assume a threshold of 2, where values <2 are 0 and others are 1:
  - Then the output pattern of activity is (1 1 0 0)

Learning Multiple Associations

- Not “computationally” surprising that an array of 24 can store the relationship between two vectors of size 6 and 4 respectively
- What happens if we try to store different associations with the same weight matrix?
  - Appearance of apricots: (1 1 0 0 0 1)
  - Taste of apricots: (0 1 0 1)

Change in weights for apricots:

		1	1	0	0	0	1
		↓	↓	↓	↓	↓	↓
0	→	0	0	0	0	0	0
1	→	1	1	0	0	0	1
0	→	0	0	0	0	0	0
1	→	1	1	0	0	0	1

The combined weight matrix:

1	0	1	0	1	0
2	1	1	0	1	1
0	0	0	0	0	0
1	1	0	0	0	1

Recall of multiple associations

- We can now see how well the pattern associator can perform recall for the 2 patterns
- Assume a threshold of 2
- Apricots:
  - Netinput: (1 4 0 3)
  - Output: (0 1 0 1)
- Chocolate:
  - Netinput: (3 4 0 1)
  - Output: (1 1 0 0)
- Both are correctly recalled

1	1	0	0	0	1		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	1
2	1	1	0	1	1	→	4
0	0	0	0	0	0	→	0
1	1	0	0	0	1	→	3

1	0	1	0	1	0		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	3
2	1	1	0	1	1	→	4
0	0	0	0	0	0	→	0
1	1	0	0	0	1	→	1

Recall, Similarity and Linear Algebra

Network behaviour can be understood in terms of vectors, matrices, and operations thereon.

- If an input pattern **a**, and the weights **w** leading from the inputs to some node are represented as vectors. Netinput to that node is the ***dot product***.
  - $\text{netinput}_i = \sum_j a_j w_{ij} = \mathbf{a} \bullet \mathbf{w}$
- If the current weights are represented by a matrix **m1**, and the change in weights by a matrix **m2**, then the new weight matrix is simply: **m1** + **m2**

Observe: the dot product is highest when two vectors are ***similar***:

- Numbers in vector 1 are similar to those in the ***corresponding positions*** in vector 2
- Thus netinput is highest for similar input/weights
- Each dissimilarity reduces the netinput
- Vectors with a dot product of 0 are said to be ***orthogonal***

Properties of Pattern Associators

Similarity in vectors:
- p: 1 0 0 0 0 1 1 1
- w1: 1 0 0 0 0 1 1 1= 4
- w2: 1 0 0 0 1 0 1 1= 3
- w3: 0 0 1 1 1 0 1 1= 2
- w4: 0 1 1 1 1 0 0 0= 0

Operation of pattern associators using the Hebb rule:

- Learning: if a neuron *i* is activated by P1, an increment Δw_i that has the same pattern as P2, is added to the weight vector of neuron *i* .

- Recall:since patterns presented during learning are directly reflected in the weight vector for neuron *i*, the output at neuron *i* reflects the similarity of the recall cue to patterns presented during learning

Properties: Generalisation, Fault tolerance, Prototype extraction, Speed

Generalisation

If a presented cue is similar to one that is learned, a pattern associator will often produce a similar response for the new as for the old pattern

- This means networks can associate “imperfect/noisy” stimuli

I.e. Insensitive to relatively small differences in input stimuli:

- E.g. (1 1 0 1 0 0) is slightly different from (1 1 0 0 0 1)
- Or, (1 0 1 0 0 0) is slightly different from (1 0 1 0 1 0)

1	1	0	1	0	0		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	1
2	1	1	0	1	1	→	3
0	0	0	0	0	0	→	0
1	1	0	0	0	1	→	2

1	0	1	0	0	0		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	2
2	1	1	0	1	1	→	3
0	0	0	0	0	0	→	0
1	1	0	0	0	1	→	1

Fault tolerance

Just as pattern associators can often deal with imperfect stimuli, they are often robust to damaged connections (synapses)

This is because PAs compute a correlation of the pattern with the weights via a relatively large number of axons

1	1	0	0	0	1		
↓	↓	↓	↓	↓	↓		
1	0	1	0	1	0	→	1
2	1	1	0	X	1	→	4
0	0	0	X	0	0	→	0
1	X	0	0	0	1	→	2

This can help explain continued (sometime partial) function in the event of normal cell loss, or certain kind of (distributed) brain damage.

More Properties

Prototype Extraction & Noise Reduction

- If the network is exposed to similar (but slightly different) P2s for a given P1 during training, the (scaled) weight vectors becomes the average P2.
- When tested, the best response is to the average pattern vector, or **prototype,** which was never explicitly seen.

Distributed Representations and Speed of Computation

- Information about the stimulus is distributed over the population of elements, rather than encoded by a single element
- Generalisation and graceful degradation rely on a continuous range of dot products
- Computation is distributed, across multiple neurons and synapses: the response to a stimulus can be determined in 1-2 steps

Summary of Pattern Associators

Associate multiple stimulus-response patterns in a single network, via a weight matrix

Weights are sensitive to similarity: The more similar, the higher the netinput; **p• w**

- Generalisation: robust to noisy input
- Fault tolerance: robust to loss/damage
- Prototype extraction & noise reduction

Biologically Plausible: Learning is strictly local, reinforcement based

Auto Association

- We can also train a network to associate a given pattern with itself
- Why? Noise reduction, prototype extraction, category formation (unsupervised)

Competitive Networks: Overview

Operation:

- Given a particular input, output units compete with each other for activation
- The winning output unit is the one with the greatest response

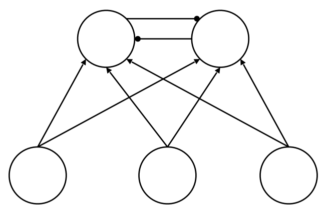
During training:

- Connections to the winning unit from the active input units are strengthened
- Connections from inactive units are weakened

Training is **unsupervised:** There is no external teacher

- The network will categorise inputs, based on similarity

Architecture of Competitive Networks



A simple network:
- Inputs are fully connected to outputs by feed-forword connections
- Outputs may be connected to each other by *inhibitory* connections

Outputs compete until only one remains active
- Or, simply the unit with highest activation wins

$$\text{netinput}_i = \sum_j a_j w_{ij}$$

Excitation of outputs:
- Dot product of input activations and the weight vector the the output

Competition:
- Output activations are compared, unit with highest activation wins
- Or, direct competition among outputs, via inhibitory connections:
  - Active units force other units to become inactive

Adjusting Weights

Weights are only adjusted on connections feeding into the winning output node:

$$\Delta w_{ij} = 0 \text{ if unit } i \text{ loses}$$
$$= \varepsilon\, (a_j - w_{ij}) \text{ if unit } i \text{ wins}$$

Where,

ε is the learning rate parameter

a_j is the activity of input unit j for pattern p

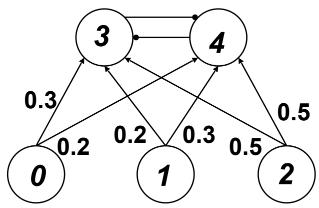
w_{ij} is the weight of the connection from j to i before the trial

The strengths of connections to the winning unit are adjusted until each weight is the same as the activity of its input

The winning unit's weight vector is changed to make is more similar to the input vector for which it is the winner

An example

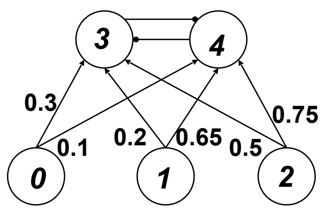
Consider the following network:



- Input pattern: (0 1 1)
  - netinput$_3$ = $(0\text{x}0.3+1\text{x}0.2+1\text{x}0.5)$ = 0.7
  - netinput$_4$ = $(0\text{x}0.2+1\text{x}0.3+1\text{x}0.5)$ = 0.8

$$\Delta w_{ij} = 0 \text{ if unit } i \text{ loses}$$
$$= \varepsilon\, (a_j - w_{ij}) \text{ if unit } i \text{ wins}$$

- Since, unit$_4$ wins:
  - No changes in connections to unit$_3$



- For connections to unit$_4$:
  - $\Delta w_{ij} = \varepsilon\, (a_j - w_{ij})$
  - $\Delta w_{ij} = 0.5\ (0–0.2\ \ 1–0.3\ \ 1–0.5)$
  - $\Delta w_{ij} = 0.5\ (–0.2\ \ 0.7\ \ 0.5)$
  - $\Delta w_{ij} = (–0.1\ \ 0.35\ \ 0.25)$

Overall behaviour

Netinput to an output unit is greatest when it's weight vector is most similar to the input vector

Training makes the weight vector for a particular winning unit more similar to the input pattern

The weight vector for a particular "winning" output unit learns to respond to similar input patterns

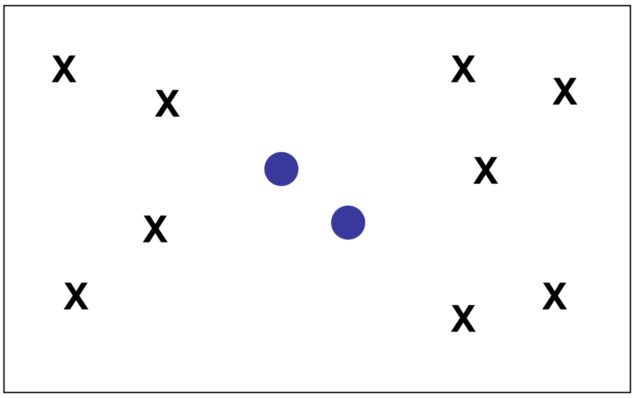
- Because these patterns are all slightly different, the learned weights cannot exactly mimic the associated inputs
- Rather, the learned weights will be an average of the patterns, based on the frequency of presentation during training

The competitive network can therefore learn to **categorise** similar inputs without any "teacher": unsupervised learning

Visualising competitive learning

Represent input patterns & weight vectors in multi-dimensional space

- weight vectors for the output units have a random relation to the input patterns
- Competitive learning changes the weight vector for a particular output so that it becomes the average for a subset of inputs
- More outputs enable the network to more finely categorise the inputs



More on competitive networks

Weight growth:

- If many similar patterns are associated with one output, it may be impossible to other outputs to ever gain more activation, even for quite different input patterns
- Can insist that the sum of a weight vector equal some constant, thus learning could only redistribute weight among connections to the winning unit

As with Hebbian networks, learning is local:

- Competition of output: inhibitory connections let only one neuron fire
- Hebbian learning means that only connection weights to this node are changed
- Information is available at the axon and dendrite of a connection
- Also: no explicit teacher is required

Remove redundancy: set of inputs is associated with a single output

- Sparsification: convert pattern stimuli to a localist representation

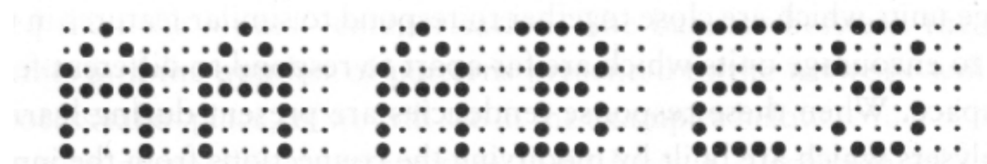
Outputs are less correlated (possibly orthogonal) than inputs:

- Useful as input to pattern associators (easier to learn less correlated patterns)

An example: Pattern classification

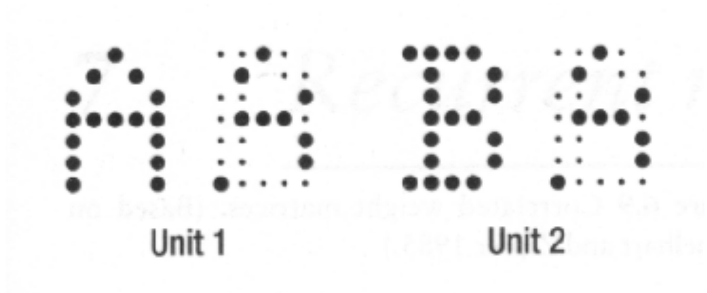
We can use an unsupervised network to classify patterns of letters

Input is a 7x14 “retina”, connected to 2 outputs via a 98 element weight vector, trained on letter pairs.



Network is first trained on: AA, AB, BA, BB. The resulting weights to the outputs are as follows:

- Unit 1: AA, AB
- Unit 2: BA, BB



Why? What else could it learn?

What would happen if the network had 4 output units?

Pattern classification continued

Consider the case where we train the network on individual letters, instead of pairs: A, B, E, S … cluster using 2 output units

The result will be to cluster A & E and B & S, since they are the most similar: thus the classifier acts as a feature detector within letters

What if the network is trained to classify AA, BA, SB, EB

- As we would expect, the network learns a letter specific classification
- But, we have forced A & B and S & E to be grouped together
- In this way, we force the network to find whatever features do ***correlate*** for the letters in the 1 position
- The 2nd letter acts as a teacher, since it forces the network into a specific solution