A small part of Lattice Theory

Ralph Selfridge

Abstract

Lattice algebras replace + and – with max and min, but then can create many of the operations of normal algebra. In particular lattice algebras can be used in the study of neural networks and extraction of images in the presence of noise. This paper is not a full introduction but a start with some of the simple algorithms that can be used.

Preliminaries

Artificial neural networks are structures designed to recall a specified pattern even in the presence of noise. Clearly this is useful in artificial intelligence and also in recovering pictures from noisy scenes. Here we are looking at the 'memories' that recover the signal, written as `M comb y`, where `M` is the 'memory' and `comb` some chosen function. While there are several possible choices we consider those from *lattice algebra*, and define `M` and `comb` accordingly.

There are two primary purposes behind this article. First is an introduction to a small subset of work within lattice theory, but second is an attempt to show that APL and J can provide proofs which in many cases are substantially shorter than conventional mathematical notation. With that in mind we provide expressions and proofs in both APL and J script, typically APL first. There are clear advantages and disadvantages for each language and we are interested in comparisons. We use caps for J nouns for ease of comparison with APL. Readers should also note that in J verb definition mode left arguments are `x`, and right arguments `y`. Many of the proofs are relegated to the appendix. All APL scripts assume `⎕IO←0`.

A lattice is a set such that any two elements have a lower bound and an upper bound. The Reals work very nicely using `max` and *min* in place of `+` and `-`. The literature on lattice theory is quite extensive, try for example a Google search on *lattice theory applications* or *lattice theory neural networks* which return quite large lists. Wikipedia, on lattice theory, appears to return heavily mathematical papers. This article has a main focus on extraction of patterns from prototypes and Ritter and Gader (ref 1) shows many such patterns.

It is fairly clear that `max` and `min` are commutative, and distributive across `+` and `-`. In particular, however, if `X` is a matrix we have

```
APL:    (⌈/⌊⌿X)≤⌊/⌈/X
  J:    (>./ <./ X) <: <./>./ "1 X
```

(In words, the maximum of the column minima is less than or equal to the minimum of the row maxima). This is called the *minimax* principle. We also remind readers that the maximum of the sum of two vectors is less than or equal to the sum of the two maxima.

We are interested in a very specific transformation of our lattice, and the question of which vectors are fixed (i.e. don't change) under that transformation. We first define a linear *minimax* combination of vectors in the array `L` by any array `R` (there must be as many rows in `R` as there are in `L`)

```
APL:  Z←L GX R
       Z←⌈⌿(⍉R)⌊.+L
  J:   GX=:>./@((<./ .+)~|:) NB.  L GX R
```

We set `G(X)` as the set of all possible linear *minimax* combinations of `X`. For example if

```
         5 5 2 0 6               1 9 2 3
X   is   3 4 3 5 4   and H  is   0 7 3 7
         5 6 0 2 1               9 1 5 9

then  8 8 5 3 9≡X GX H
```

If `Y` can be written as `X GX H` for some `H`, `Y` is said to be lattice dependent on `X`.

We wish to define two memories, clearly dependent on the vectors given by the rows of `X`. With these memories we have a transformation, where the memory is always the left argument, applied to some other vector.

We now define two ‘memories’ (the first line in the following code) and their associated functions or ‘transformations’ (the next two lines in APL, the second line in J):

```
APL :  M←(⍉R)⌈.-R          W←(⍉R)⌊.-R
       Z←M MT R            Z←W WT R
       Z←R⌊.+⍉M            Z←R⌈.+⍉W
  J :  M=:((>:/ .−)~|:)y   W=: ((<./ .−)~|:)y
       MT =: (<./ .+|:)~   WT =: (>./ .+|:)~
```

NB. Typically M&MT or W&WT

and ask when Y≡M MT Y (Y-: M MT Y)) or Y≡W WT Y (Y-:W WT Y), i.e. is Y a fixed point for the transformation M MT (M&MT) or W WT (W&WT)? The set of fixed points of either transformation is F(X) (fixed under M MT if and only if fixed under W WT, see Theorem 1).

It is trivial to show that the diagonals of M and W are all zero, and that M is the negation of the transpose of W. This allows the easy proof of:

Lemma 1:

```
APL:  (M MT Y)≤Y+ 0 0⍉M   ≡Y
      (and Y≤W WT Y)
  J:  (M MT Y)< : Y + (<0 1)|:M ↔ Y,
      and similarly Y<: W WT Y
```

Lemma 2:

```
APL:  M≡M⌊.+M
  J:  M–: (<./ .+)~M
```

See proof in the appendix.

Theorem 1:

Y≡W WT Y if and only if Y≡M MT Y (i.e. Y is fixed under W WT if and only if Y is fixed under M MT).

See proof in the appendix.

Theorem 2:

The vectors of X are all in F(X).

See proof in the appendix.

Theorem 3:

Y ∊ F(X) ↔ Y is lattice dependent on X.

See proof in the appendix.

Corollary:

If Y is lattice dependent on X then H←Y-[1]X (H=.Y − "1 X) is suitable for Y≡X GX H (Y −: X GX H). H is not unique. (This corollary is an emphasis of the first line of the proof).

We want to know more about the shape of F(X), and ultimately its dimension and extreme points (which are, in fact, lines). We start by showing that F(X) is a convex set.

The fact that the diagonal of M is all zero allows us to say, for any p in [0,1], that if R,S∊F(X) then

```
   M MT(p×R)+(1-p)×S
≡  ((p×M)MT p×R)
   +((1-p)×M)MT(1-p)×S
≡  (p×M MT R)+(1-p)×M MT S
≡  (p×R)+(1-p)×S
```

Thus F(X) is a convex set.

If we consider X, adding a constant to any vector will not change M. Similarly if R∊F(X) then R+c∊F(X), for any c. Thus X generates a convex volume F(X) in R^n , and if R∊F(X) the line R+c is in F(X) (effectively R represents the bundle of lines going through point R and parallel to x=y=z...)

A vector V is said to be lattice independent of X if V is not a lattice combination of the vectors in X, or, equivalently, V is not in F(X). A set X is said to be lattice independent if no vector in X is lattice dependent on the remaining vectors (V∊X ↔ V is not in F(X\V).

In order to proceed further we need the idea of strong lattice independence motivated by the following:

Theorem 4:

If R and S are lattice independent then there exist indices p and q such that for all i (R[p]-R[i])≤S[p]-S[i] and (S[q]-S[i])≤R[q]-R[i].

The contra-positive for the first choice is that for any index j[k] there is a next index j[k+1] such that (strict inequality):

```
(R[j[k]]-S[j[k]])<
R[j[k+1]]-S[j[k+1]]
```

Now the sequence of indices is infinite but there is only a finite possible set and we have proof by contradiction (the proof for J is clear).

This is the background for:

Definition:

A set *X* of lattice independent vectors is strongly lattice independent if and only if: for every *R*∈*X* there is an index *k* such that for any pair of indices *i* , *j* :

```
APL: (R[k]-R[i])≥X[j;k]-X[j;i]
   J:   ((k{R)-i{R)>: ((<j,k){X)-(<j,i){X
```

We call this inequality, by itself, condition *S* which can be used without lattice independence (which, of course, prevents strong lattice independence).

We now wish to find the strongly lattice independent vectors in *F(X)* and then the minimal set which generates *F(X)*. Remember that *X* may not even be lattice independent (but if there are only 2 different vectors in *X* they will be strongly lattice independent).

Theorem 5:

The negated rows of *M* all satisfy condition *S*, and are lattice dependent on *X* (specifically we do not claim these rows are lattice independent against each other).

Condition S is easily satisfied by the negated rows since

```
    (X[i;k]-X[i;j])≤M[k;j]
≡  (X[i;k]-X[i;j])≤X[k;k]-X[k;j]
```

In J we have:

```
    (-/ (i,j ; i,n){X)<:(<j,n){M
←→ (-/ (i,j ; i,n){X))<: -/(j,j ; j,n){X
```

which is condition *S* with k=.j.

The proof of lattice dependence on *X* is in the appendix.

It is now clear that if a vector *y* satisfies condition *S* it must be one of the negated rows of *M*, with a possible added constant, and we have *F(X)≡F(M)*. Now we are interested in the minimal set of strongly lattice independent vectors, *Q*, with *F(Q) ≡F(X)*; we call this the minimal spanning set. Such a spanning set is made up of lattice independent vectors from *-M* and algorithm *A* provides for finding such a set.

Algorithm A:

Given a defining set *X*, set *H*←-(⍉*X*)⌈.-*X* (H=.-((>./ .-)~|:)X) (i.e. the negated rows of the memory generated by X). We build the spanning set *Q* by removing any vector in *H* that does not change *M*. The remaining vectors, after all have been examined, form the set *Q*.

Clearly we have F(X)≡F(Q), and the remaining vectors must be lattice independent, so *Q* is a minimal spanning set (but not necessarily unique). We provide *FIND* in APL and J.

```
APL:    H←FIND R;I;M;V
        I←0◊H←-M←(⍉R)⌈.-R
        →(I=↑ρH)/0
        ⍎(M≡(⍉V)⌈.-V←(I≠ι↑ρH)⌿H)/
              'H←V◊→2'
        I←I+1◊→2

 J:     FIND=: 3 : 0
        NB.  y is the right argument in definition mode
        H=.-M=.((>./ .-)~|:)) y
        i=.0
        while. i<#W  do.
        if.M-:((>./ .-)~|:)V=.(i~:i.#H)#H
              do.H=.V
        else. i=.>:i  end. end.
        H
        )
```

For example if we use the array *X* from above we have

```
        X              M              FIND X
5  5  2  0  6    0  0  5  5  4    ¯1  0¯6¯5¯5
3  4  3  5  4    1  0  6  5  5    ¯2¯1¯2  0¯1
5  6  0  2  1    0¯1  0  2¯1     ¯1¯1¯4¯6  0
                 2  1  2  0  1
                 1  1  4  6  0
```

Strong lattice independence provides some level of access to linear independence and efficient ways of handling lattice algebras.

We can, in fact, show the spanning set is unique, but the proof is awkward, wandering around in R^n.

It is possible to show that if we take the set M, and remove one row, while subtracting it from all the others, then the remaining vectors are linearly independent (a moderately difficult proof). We also have:

Conjecture:
M is a set of linearly independent vectors so long as no row or column is all zero. This conjecture comes from being unable, so far, to find a linearly dependent set except by getting a row or column all zero. Obtaining a row or column all zero has, to date, meant a row and column of X being all zero and all entries being non-negative or non-positive (but proving the conjecture has not yet been accomplished).

The structure we have built, starting from a set of vectors X, and constructing associative memories and minimal spanning sets has value in several areas of study, including neural nets and recovery of pictures from noisy pictures. We provide an example after the references where the start is a set of letters, and then recovery from distortions of some of those letters.

We add that, except for the J and APL scripts, only portions of this article are original to the author, much of it can be found in the few references provided.

References

G.X.Ritter, Paul Gader, Fixed points of Lattice Transforms and Lattice Associative Memories; Advances in Imaging and Electron Physics 144 (2006) 165-242

G.X.Ritter, P.Sussner, J.L. Diaz de Leon, Morphologiccal Associative Memories, IEEE Trans on Neural Networks, 9(2)1998, p 281-293

G.Birkhoff, Lattice Theory, American Mathematical Society Providence RI 1984

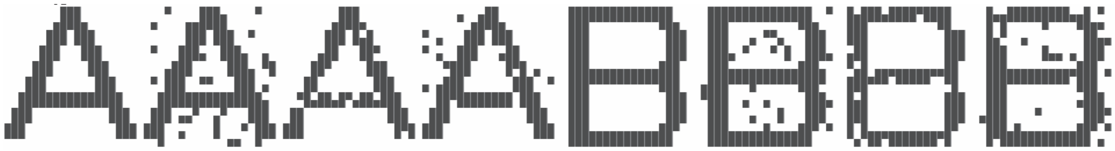
J.J.Hopfield, D.W.Tank Computing with neural circuits, Science 233(1986) p 625-633

M.A.Arbib (ed), Handbook of Brain Theory and Neural Networks, MIT Press, 1998

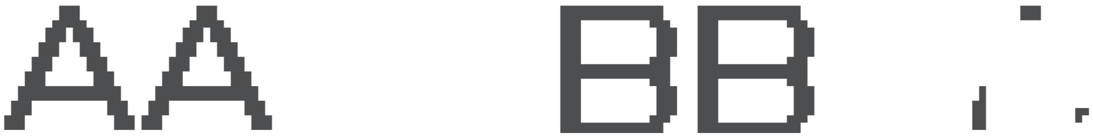
Example of extracting pictures

If we start with a training set of pictures (in this example below we use A, B, C, E) each can generate a bit vector (i.e. pixels on or off). This set of vectors then generates M. We now use M MT Yd where Yd is a dilated member Y of the training set, i.e. one of the letters with added ‘on’ pixels. The true Y is recovered, sometimes with a surprising amount of dilation. In a similar fashion we can use W WT and recover the correct Y after a surprising amount of erosion, that is, we turn a number of pixels from ‘on’ to ‘off’. Unfortunately a combination of dilation and erosion gets very poor results. Below we have A and B as defined, followed by dilation, erosion and a combination. Dilation or erosion changed approximately 15% of the available pixels in the example.

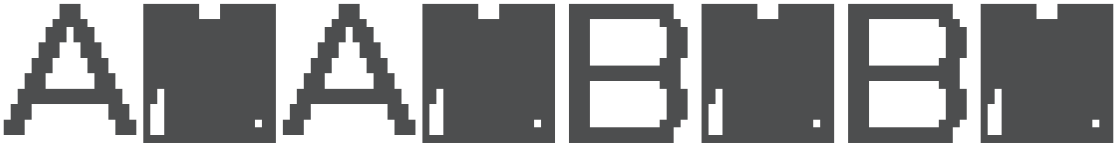
The letters A and B are presented in next image, first the original, then dilated, then eroded, then a mixture of both dilation and erosion.



Next image has the results of M MT on each picture, erosion and 'both' are useless.



Next image has the results of W WT on each picture, dilation and 'both' are useless.



Appendix: Proofs of some statements

Lemma 2

APL: For indices I and J we have
M[I;J]≡M[J;J]+M[I;J] ≥ M[;I]⌊.+M[;J]
≡ ⌊/(⌈⌿X[;I]-[0]X)+⌈⌿X-[0]X[;J] ≥ ⌊/⌈⌿X[;I]-X+X-X[;J]
≡ M[I;J] and all inequalities must be equalities.

J: For indices i and j we have (<i,j){M -: ((<i,j){M) + (<j,j){M >: <./ (i{M)+j{"1 M
←→ <./ (>./(i{"1 X) -X)+> ./X- j{"1 X >: <./>./ ((i{"1 X) -X)+X-j{"1 X)
←→ (<i,j){M and all inequalities must be equalities.

Theorem 1

APL: For any indices I and J and Y≡W WT Y
Y[I]≡Y⌈.+W[I;]
≡ Y[I]≥Y[J]+W[I;J] for any J
≡ W[I;J]≤Y[I]-Y[J]
Now if ~Y≡M MT Y then for some S we have for all K
Y[S]>Y⌊.+M[S;]
≡ Y[S]>Y[K]+M[S;K]
≡ (Y[K]-Y[S])<-M[S;K]
≡ (Y[K]-Y[S])<W[K;S]≤Y[K]-Y[S] a contradiction.

J: If Y-:W WT Y then for any indices i and j
(i{Y)-:Y>./ .+i{W
←→ (i{Y)>:(j{Y)+(<i,j){W for any j
←→ ((<i,j){W)< (i{Y)-j{Y
Now if -.Y-:M MT Y then for some s and all k we have

(s{Y)>Y<./ .+s{M
↔ (s{Y)>(k{Y)+(<s,k){M
↔ ((k{Y)-s{Y)<-(<k,s){M
↔ ((k{Y)-s{Y)<(<k,s){W
but we have ((<k,s){W) <: (k{Y)-s{Y and again a contradiction.

The reverse from M MT to W WT is similar.

Theorem 2

APL: If *V* is one of the vectors of *X* we have
W WT V≡*V*⌊.+⍉*W* ≡*V*⌈.-*M*
From lemma 1 we have *V*≤*W WT V*, so let *J* be an index for which we have a strict inequality, or *V*[*J*]<*V*⌈.-*M*[;*J*]. Thus for some *K*
V[*J*]<*V*[*K*]-*M*[*K*;*J*]
≡*M*[*K*;*J*]<*V*[*K*]-*V*[*J*] which contradicts the definition of *M*.

J: If V is one of the vectors of X we have
W WT V ↔ V<./ .+|: W ↔ V>./ .-M
From lemma 1 we have V<: W WT V, so consider the index j for which this is a strict inequality, or (j{V)>./ .-j{"1 M

If **V** is one of the vectors of **X** we have
W WT V ≡ V<./ .+|:W ≡ V>./ .-M
From lemma 1 we have **V<: W WT V**
Consider the index j for which this is a strict inequality, or **(j{V)<V>./ .- j{ "1 M**
Thus for some **k** we have
(j{V)<(k{V)-(<k,j){M ≡ ((<k,j){M)<k{V)-j{V
which contradicts the definition of **M.**

Theorem 3

APL: Set *H*←*Y*-[1]*X* then *X GX H*≡⌈/(⍉*H*)⌊.+*X*
≡ ⌈/(*Y*-⍉*X*)⌊.+*X*
≡ ⌈/*Y*-(⍉*X*)⌊.+*X*
≡ ⌈/*Y*-*M*
≡ *Y*⌈.+⍉*W*
≡ *W WT Y*≡*Y*

J: Set H=.Y-"1 X then GX H ↔ >./(|: H)<./ .+X
↔ >.(Y-(|: X)<./ .+X
↔ >./Y-M ↔ Y>./ .+|: W ↔ W WT Y ↔Y