A simple application of Artificial Neural Network to cloud classification

Tianle Yuan

For AOSC 630 (by Prof. Kalnay)

Introduction to Pattern Recognition (PR)

- Example1: visual separation between the character ‘M’ and ‘N’
- Example2: tell an elephant from a crocodile in a picture
- Example3: identification of someone’s voice from an piece of audio stream

…

The problem

- They all sound easy from the perspective of a brain.
- However, if all the information is given as a matrix, to ask a machine to finish the job is non-trivial.
- The problem: how to numerically categorize the information

Two steps

Step One:

Feature extraction (typically require human intervention, creative step)

Step two:

Categorization (typically automatic and largely numerical, technical step)

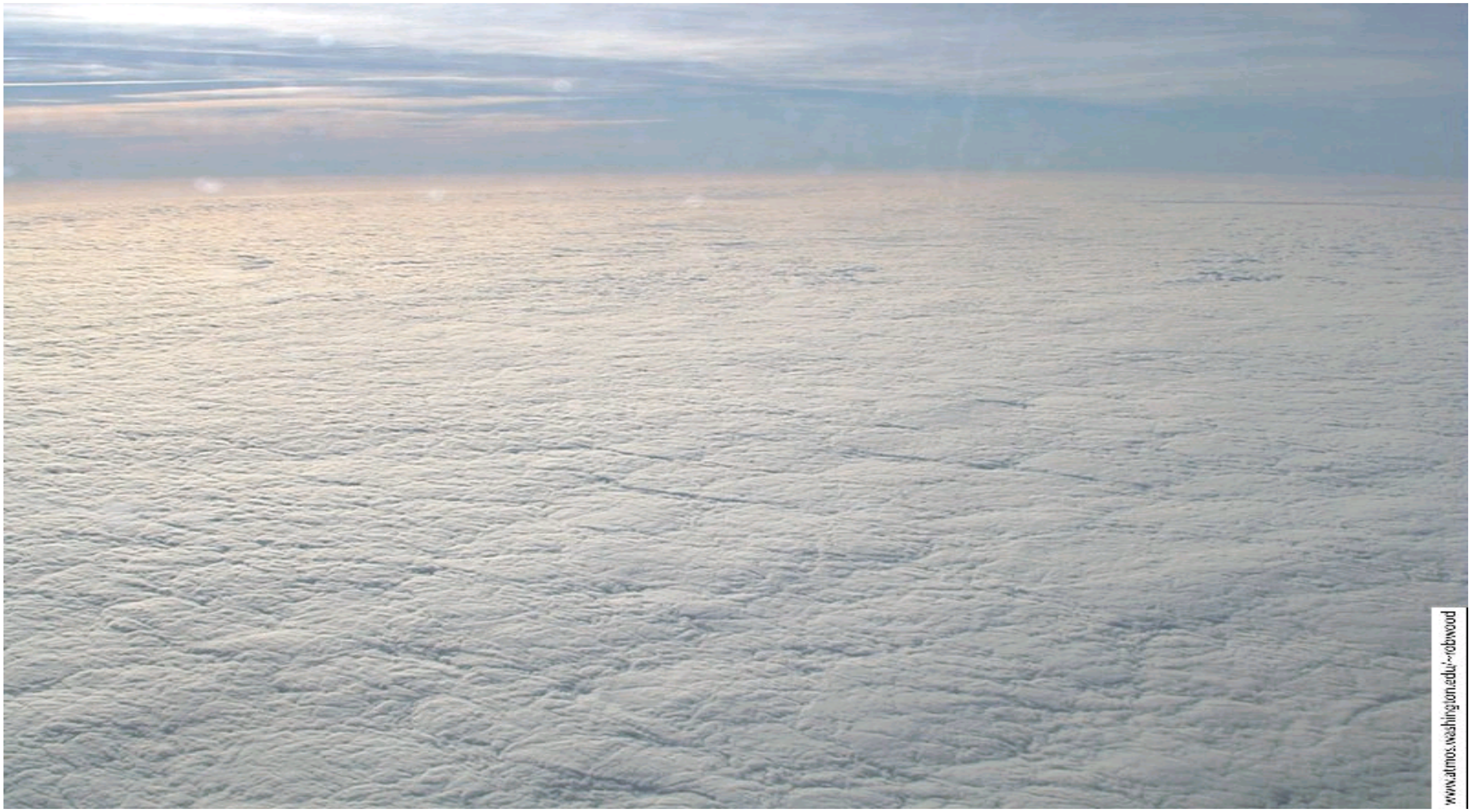
Brief Review of ANN

- ANN is basically a mapping tool for complicated input-output relationships.
- It’s made up of layers of neurons. Each neuron is a transformation function.
- The most important step is training which involves minimization of a cost function.
- Once training is finished and validated, the application is cheap and fast.

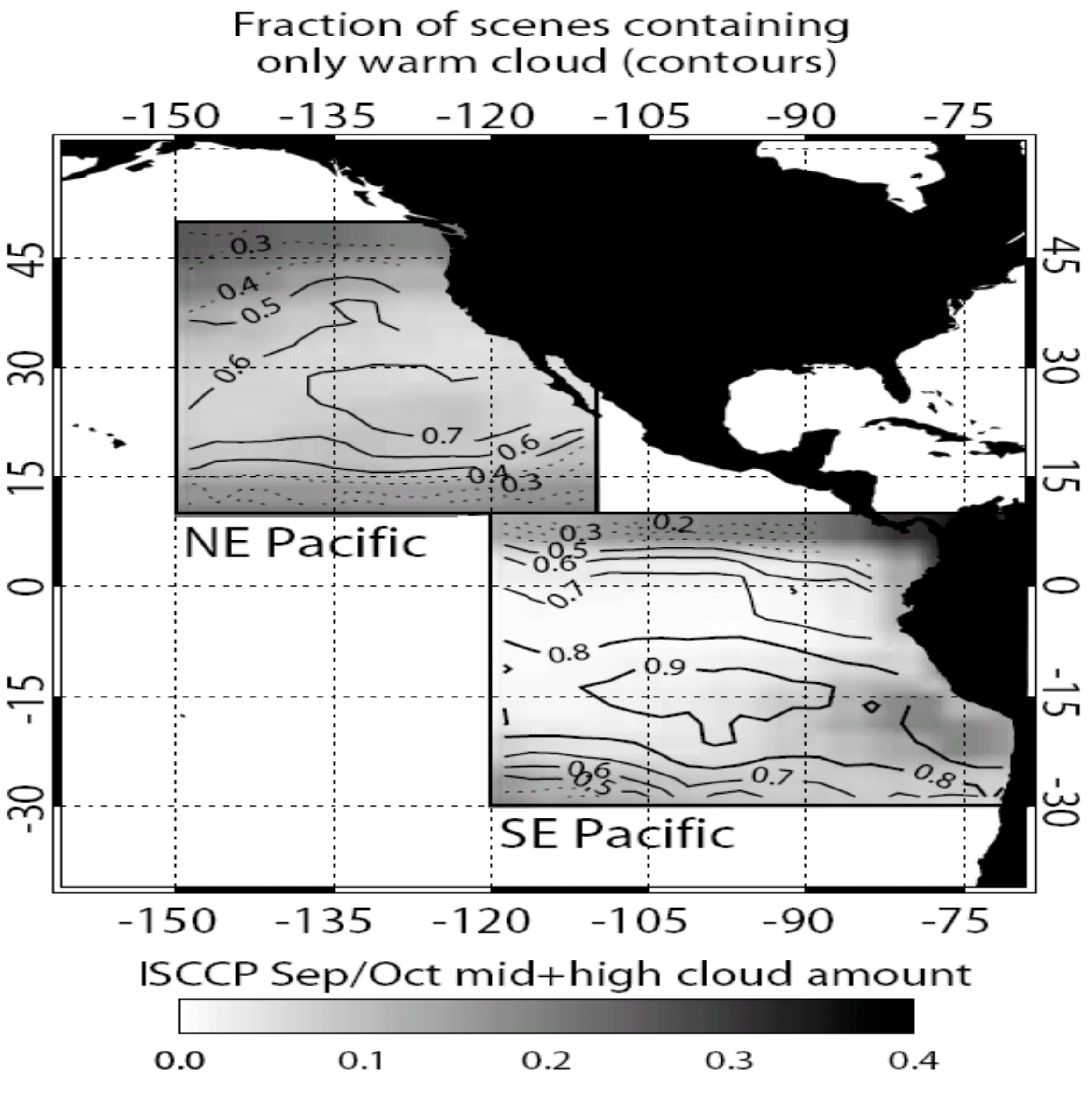
Apply ANN to Pattern Recognition

- Extract feature vectors from patterns
- Train the ANN
  - Try different initial guesses
  - If not successful try different feature vectors
  - Error tolerance
- Validate the ANN with independent samples
  - Large enough independent sample
  - Success rate
- Application

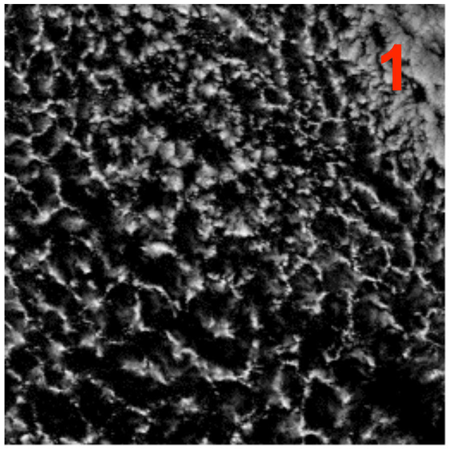
Stratocumulus Clouds

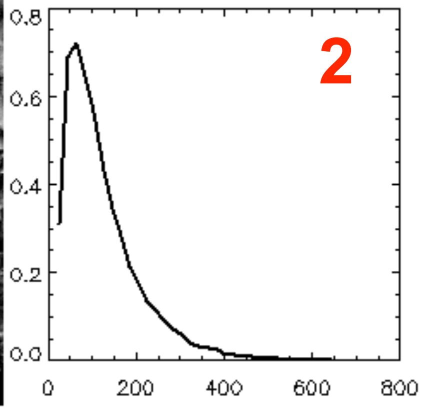


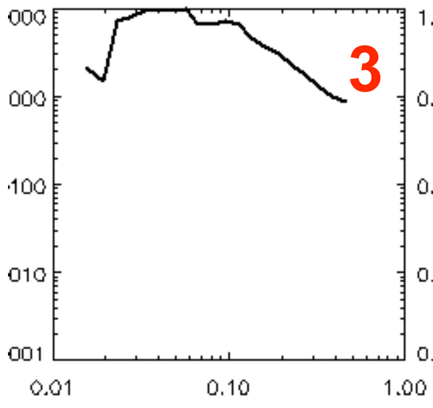
Climatically Important

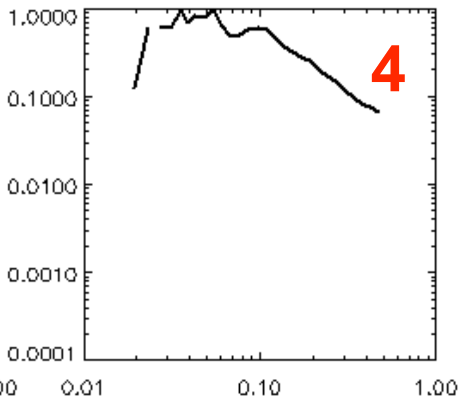


Statistics of a Scene







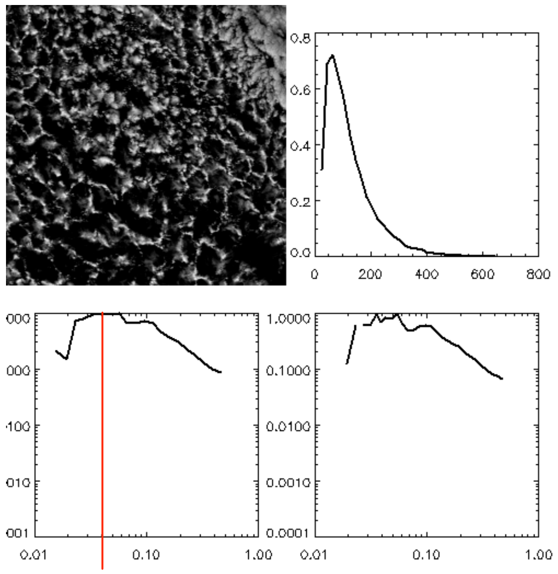


Panel 1: A 1-km solution cloud reflectance picture with size of 256X256. Cloud properties like cloud optical depth and cloud liquid water path can be derived.

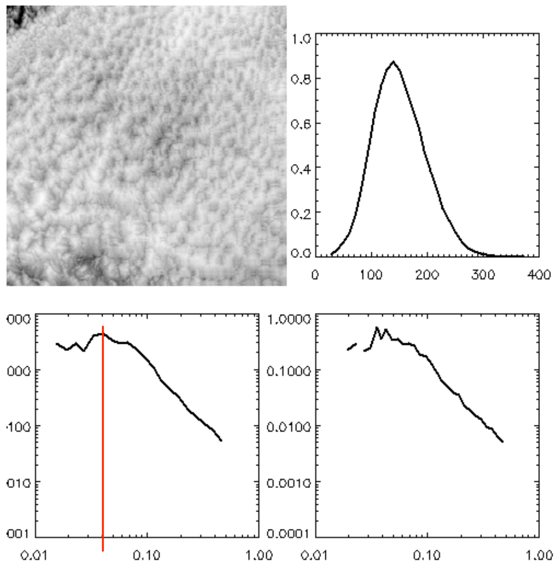
Panel 2: Liquid water path distribution

Panel 3&4: power spectra of liquid water path of image 1. panel 3 has 30 bins and panel 4 has 40 bins in the x axis (wave number)

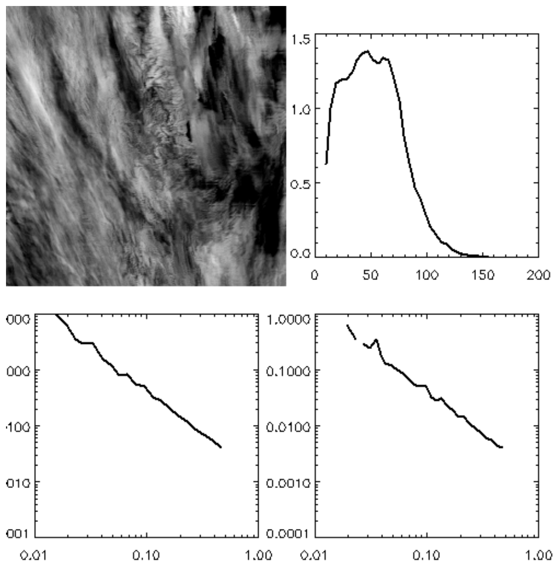
Cloud scenes



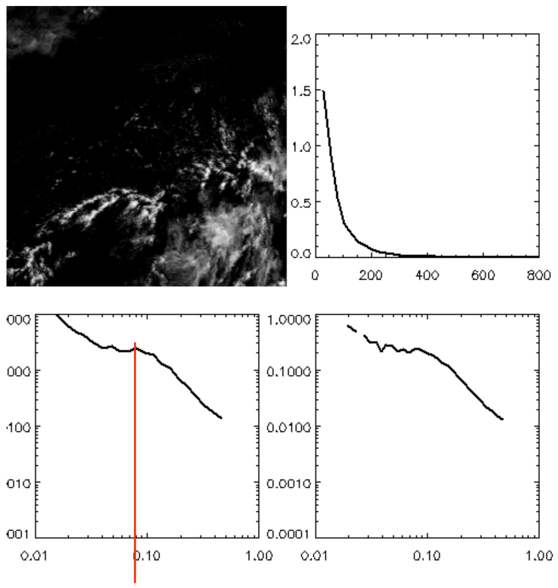
a



b



c



d

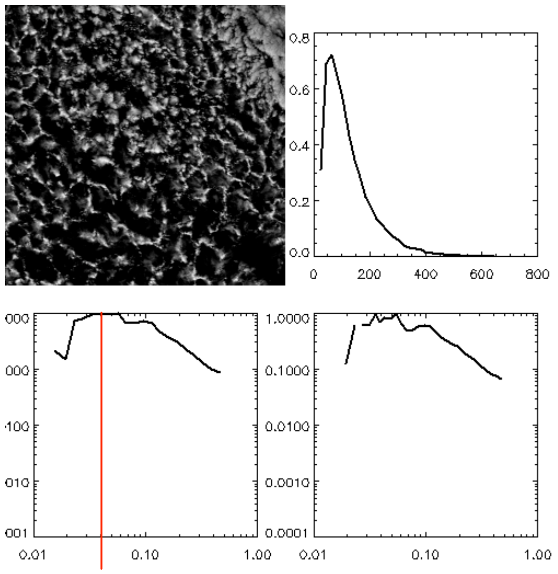
What do we know

- Type a: open cellular
- Type b: closed cellular
- Type c: non-structural, uniform
- Type d: cellular, individually organized

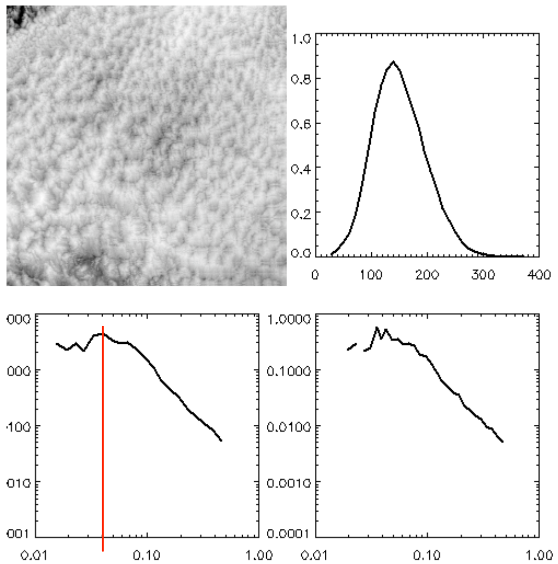
What do we know

- Liquid water path distributions for types a and d are distinct from types b and c. One has a modal distribution and the other is non-modal.
- Spatial organizations of these cloud types are also distinct. Types a and b both have a characteristic scale while c and d hardly show any regularity.

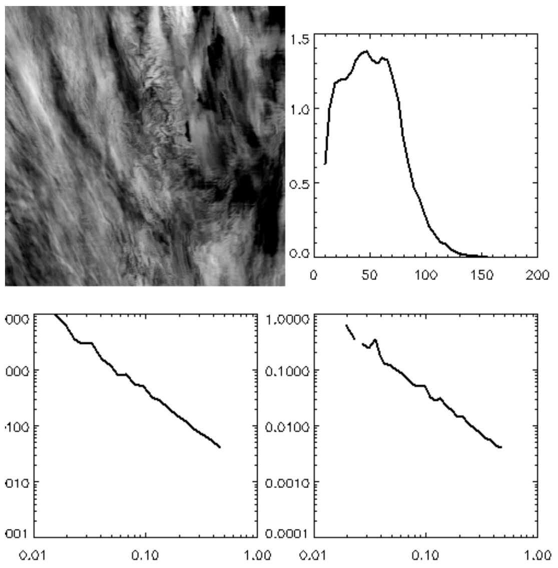
Cloud scenes



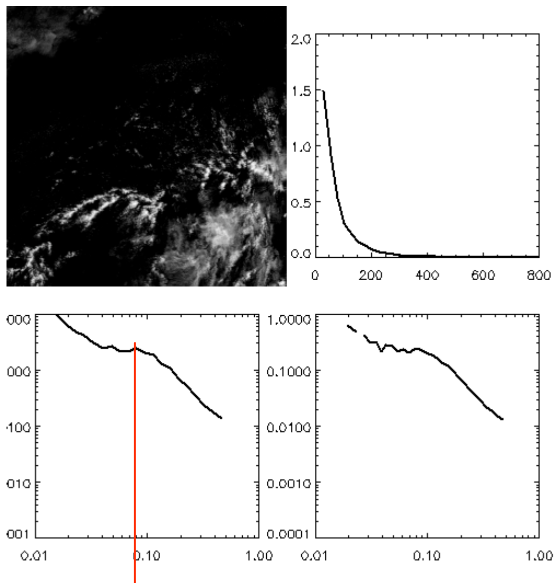
a



b



c



d

Ideas

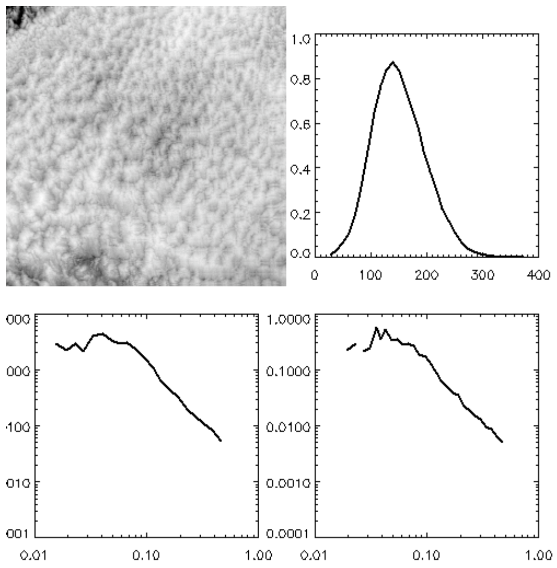
- Liquid Water Path (LWP) probability distribution (modal vs. non-modal)
- Use power spectrum of LWP for each cloud scene to represent spatial structure because power spectrum tells us which spatial frequency is prominent in images. Non structural images shall display a simple power law in the power spectrum.

Feature vector (input)

FV = [a1,…,a30,b1,…,b40], where a1-a30 are normalized frequencies of 30 LWP bins for a cloud scene; b1-b40 present normalized power spectrum of a cloud scene using 40 bins.

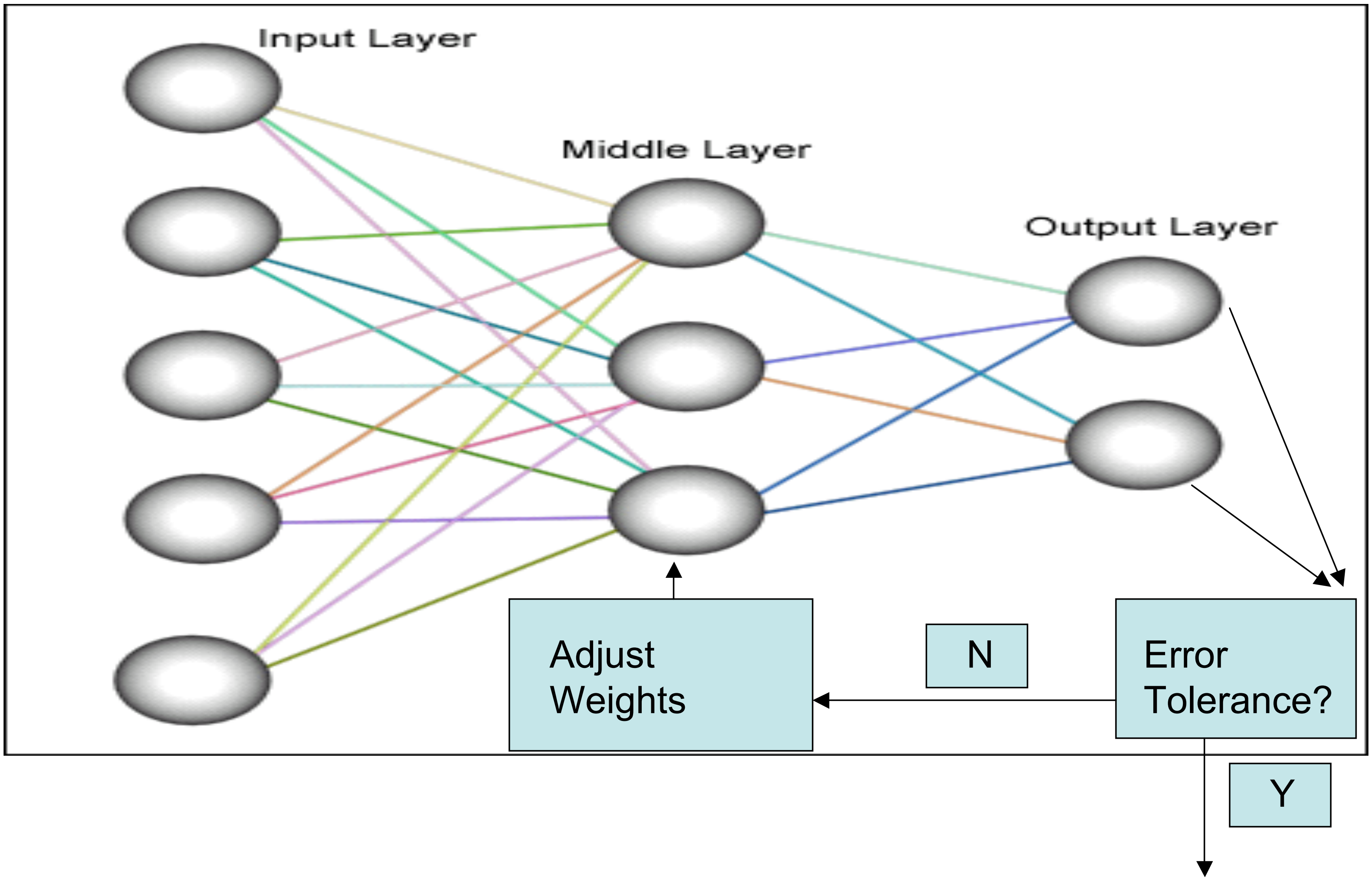
So FV is a 70-element vector that includes both LWP probability distribution and image spatial structure information.

One may modify the number of bins to present both distributions to test the sensitivity of ANN.



Classification into a, b, c, d (output)

Structure of the ANN



Training and validation

- Humanly inspect about eight hundred cloud scenes and categorize them.
- Randomly pick about 400 scenes with each category having 100 samples.
- Train the ANN until it performs as desired and apply to validation set (the rest of the 400 scenes).

Results (Accuracy)

- Training set: correctly picked 378 out of 380 scenes.
- Validation set: correctly identified 356 out of 381 scenes
- Sensitivity tests were performed with respect to the lengths of LWP PDF and power spectrum vectors, and. Results are similar.
- Most miss-identifications occur for cloud type d, as shown by the following table (proportion correct=Accuracy)

Type	a	b	c	d
T.A.	1.0	0.994	1.0	0.95
V.A.	0.925	0.933	0.986	0.75

How the algorithm works

- First, do a forward calculation and get the error
- Then adjust the weights and biases of output neurons.
- Since we know the function form of the transformation function, back propagation of error is then carried out and weights and biases of hidden layer is adjusted.
- Check the accuracy and if not satisfied go to step one again.

Codes

- They are written in IDL with comments explaining the usage and how to read the program with basic understanding of ANN.
- One can create one’s own normalized spectrum data and use this ANN to train and classify data patterns.

What’s inside the codes?

- A transformation function, in my case it’s a sigmoid function $1/(1+\exp(-x))$, mapping inputs from first layer to hidden layer outputs and then to output layer.
- An iteration algorithm that calculates error, back propagates and adjusts weights and biases in order to minimize errors.

“Neurons”

```
; compute values of hidden layer
neurons (z)

z_in =
transpose(input#w_hid+bias_hid)
z = 1.0d0/(1+exp(-z_in))
```

“z_in”: the input feature vector for the hidden layer; “z” is the out put from the layer

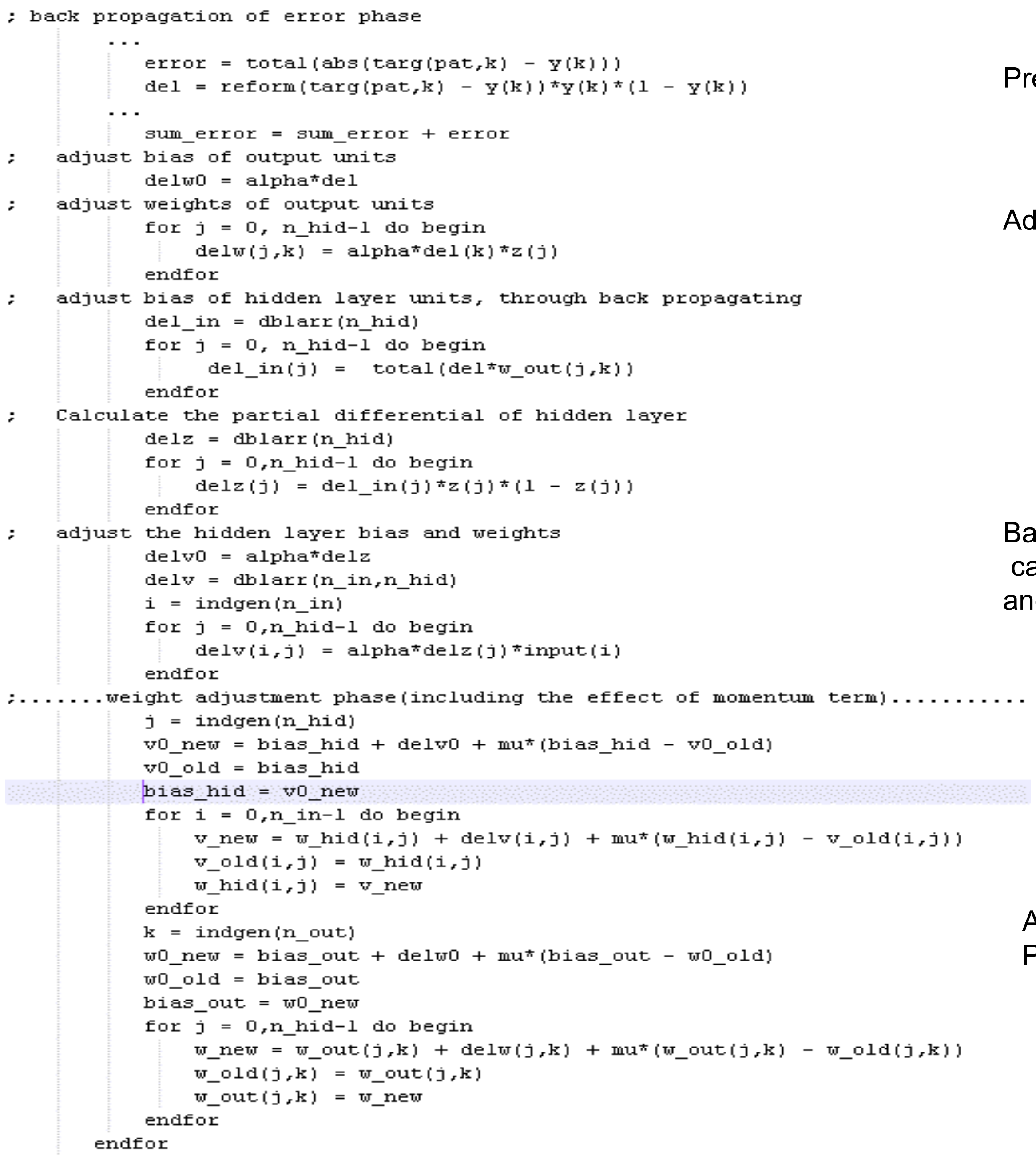
```
; compute values of output layer
neurons (y)

y_in = transpose(z#w_out+bias_out)
output = 1.0d0/(1+exp(-y_in))
h_output = z
```

“y_in”: the input vector for the output layer; “output” is the out put from the layer

back

Essential code for the iteration



```
; back propagation of error phase
        ...
           error = total(abs(targ(pat,k) - y(k)))
           del = reform(targ(pat,k) - y(k))*y(k)*(1 - y(k))
        ...
           sum_error = sum_error + error
;   adjust bias of output units
           delw0 = alpha*del
;   adjust weights of output units
           for j = 0, n_hid-1 do begin
               delw(j,k) = alpha*del(k)*z(j)
           endfor
;   adjust bias of hidden layer units, through back propagating
           del_in = dblarr(n_hid)
           for j = 0, n_hid-1 do begin
                del_in(j) =  total(del*w_out(j,k))
           endfor
;   Calculate the partial differential of hidden layer
           delz = dblarr(n_hid)
           for j = 0,n_hid-1 do begin
               delz(j) = del_in(j)*z(j)*(1 - z(j))
           endfor
;   adjust the hidden layer bias and weights
           delv0 = alpha*delz
           delv = dblarr(n_in,n_hid)
           i = indgen(n_in)
           for j = 0,n_hid-1 do begin
               delv(i,j) = alpha*delz(j)*input(i)
           endfor
;.......weight adjustment phase(including the effect of momentum term)..........
           j = indgen(n_hid)
           v0_new = bias_hid + delv0 + mu*(bias_hid - v0_old)
           v0_old = bias_hid
           bias_hid = v0_new
           for i = 0,n_in-1 do begin
               v_new = w_hid(i,j) + delv(i,j) + mu*(w_hid(i,j) - v_old(i,j))
               v_old(i,j) = w_hid(i,j)
               w_hid(i,j) = v_new
           endfor
           k = indgen(n_out)
           w0_new = bias_out + delw0 + mu*(bias_out - w0_old)
           w0_old = bias_out
           bias_out = w0_new
           for j = 0,n_hid-1 do begin
               w_new = w_out(j,k) + delw(j,k) + mu*(w_out(j,k) - w_old(j,k))
               w_old(j,k) = w_out(j,k)
               w_out(j,k) = w_new
           endfor
       endfor
```

Predicted minus Observed cloud class

Adjust weights and biases of output layer

Based on the Sigmoid function analytically calculate differential and adjust weights and biases

Add some momentum term to the adjusting Process to get fast convergence

back