



**Defense Special Weapons Agency  
Alexandria, VA 22310-3398**



**DSWA-TR-97-82**

## **Worldwide Cloud Prediction Algorithms**

**Kenneth A. Poehls, et al.  
Pacific-Sierra Research Corp.  
2901 28th Street  
Santa Monica, CA 90405-2938**

**November 1998**

**Technical Report**

**CONTRACT No. DNA 001-94-C-0149**

**Approved for public release;  
distribution is unlimited.**

**19990119 092**

DESTRUCTION NOTICE:

Destroy this report when it is no longer needed.  
Do not return to sender.

PLEASE NOTIFY THE DEFENSE SPECIAL WEAPONS  
AGENCY, ATTN: CSTI, 6801 TELEGRAPH ROAD,  
ALEXANDRIA, VA 22310-3398, IF YOUR ADDRESS IS  
INCORRECT, IF YOU WISH IT DELETED FROM THE  
DISTRIBUTION LIST, OR IF THE ADDRESSEE IS NO  
LONGER EMPLOYED BY YOUR ORGANIZATION.

# REPORT DOCUMENTATION PAGE

*Form Approved*  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204 Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 981101	3. REPORT TYPE AND DATES COVERED Technical 940826 — 970430	
4. TITLE AND SUBTITLE Worldwide Cloud Prediction Algorithms		5. FUNDING NUMBERS C - DNA 001-94-C-0149 PE - 62715H PR - AC TA - BA WU - DH00114	
6. AUTHOR(S) Kenneth A. Poehls, David M. Crandall, Kevin O'Rourke, and Kenneth E. Heikes		8. PERFORMING ORGANIZATION REPORT NUMBER  PSR Report 2732	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Pacific-Sierra Research Corp. 2901 28th Street Santa Monica, CA 90405-2938		10. SPONSORING /MONITORING AGENCY REPORT NUMBER  DSWA-TR-97-82	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESSEE(S) Defense Special Weapons Agency 6801 Telegraph Road Alexandria, VA 22310-3398 WEL/Smith		11. SUPPLEMENTARY NOTES This work was sponsored by the Defense Special Weapons Agency Under RDT&E RMC Code B4662D BA 00114 4400A AC 25904D.	
12a. DISTRIBUTION/AVAILABILITY  Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT ( <i>Maximum 200 words</i> )  This report describes the major algorithms included in the Worldwide Cloud Prediction Model (WCPM). A lack of data supplied by DSWA precluded code development beyond a feasibility level. Algorithm performance is presented in the project final report (Poehls, Crandall, O'Rourke and Heikes; 1997).  The forecast is designed around a unified neural network with weather inputs representing <i>advection</i> , <i>persistence</i> , and <i>evolution</i> of clouds. Cloud observation data is taken from SERCAA level 3 nephanalysis. NOGAPS forecasts are used for the meteorological parameter inputs. The adopted pixel-by-pixel approach assumes that a forecast is possible based solely upon the past, current and approaching clouds. Each pixel is only loosely connected to surrounding pixels through geographic inputs. No formal synoptic weather inputs are employed.			
14. SUBJECT TERMS Weather                      Neural Networks                      Cloud Forecasting                      SERCAA			15. NUMBER OF PAGES 68
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

CLASSIFIED BY:

N/A since Unclassified.

DECLASSIFY ON:

N/A since Unclassified.

13. ABSTRACT (Continued)

The major pieces are the neural network itself and the advection algorithm utilized to locate data in space and time. All other algorithms provide either neural network input or training data. The general form, the training process, and the final input vectors to the neural network are detailed. The persistence and evolution algorithms actually represent the final input choices for specific space-time data. Although separately described, there was never any intention that the algorithms would perform as stand alone modules.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

## CONVERSION TABLE

Conversion factors for U.S. Customary to metric (SI) units of measurement

MULTIPLY TO GET	→ ←	BY BY	→ ←	TO GET DIVIDE
angstrom		1.000 000 x E -10		meters (m)
atmosphere (normal)		1.013 25 x E +2		kilo pascal (kPa)
bar		1.000 000 x E +2		kilo pascal (kPa)
barn		1.000 000 x E -28		meter <sup>2</sup> (m <sup>2</sup> )
British thermal unit (thermochemical)		1.054 350 x E +3		joule (J)
calorie (thermochemical)		4.184 000		joule (J)
cal (thermochemical)/cm <sup>2</sup>		4.184 000 x E -2		mega joule/m <sup>2</sup> (MJ/m <sup>2</sup> )
curie		3.700 000 x E +1		*giga becquerel (GBq)
degree (angle)		1.745 329 x E -2		radian(rad)
degree Fahrenheit		$t_K = (t_F + 459.67)/1.8$		degree kelvin (K)
electron volt		1.602 19 x E -19		joule (J)
erg		1.000 000 x E -7		joule (J)
erg/second		1.000 000 x E -7		watt (W)
foot		3.048 000 x E -1		meter (m)
foot-pound-force		1.355 818		joule (J)
gallon (U.S. liquid)		3.785 412 x E -3		meter <sup>3</sup> (m <sup>3</sup> )
inch		2.540 000 x E -2		meter (m)
jerk		1.000 000 x E +9		joule (J)
joule/kilogram (J/kg) (radiation dose absorbed)		1.000 000		Gray (Gy)
kilotons		4.183		terajoules
kip (1000 lbf)		4.448 222 x E +3		newton (N)
kip/inch <sup>2</sup> (ksi)		6.894 757 x E +3		kilo pascal (kPa)
ktap		1.000 000 x E +2		newton-second/m <sup>2</sup> (N-s/m <sup>2</sup> )
micron		1.000 000 x E -6		meter (m)
mil		2.540 000 x E -5		meter (m)
mile (international)		1.609 344 x E +3		meter (m)
ounce		2.834 952 x E -2		kilogram (kg)
pound-force (lbs avoirdupois)		4.448 222		newton (N)
pound-force inch		1.129 848 x E -1		newton-meter (N m)
pound-force/inch		1.751 268 x E +2		newton/meter (N/m)
pound-force/foot <sup>2</sup>		4.788 026 x E -2		kilo pascal (kPa)
pound-force/inch <sup>2</sup> (psi)		6.894 757		kilo pascal (kPa)
pound-mass (lbm avoirdupois)		4.535 924 x E -1		kilogram (kg)
pound-mass-foot <sup>2</sup> (moment of inertia)		4.214 011 x E -2		kilogram-meter <sup>2</sup> (kg m <sup>2</sup> )
pound-mass/foot <sup>3</sup>		1.601 846 x E +1		kilogram/meter <sup>3</sup> (kg/m <sup>3</sup> )
rad (radiation dose absorbed)		1.000 000 x E -2		**Gray (Gy)
roentgen		2.579 760 x E -4		coulomb/kilogram (C/kg)
shake		1.000 000 x E -8		second (s)
slug		1.459 390 x E +1		kilogram (k)
torr (mm Hg, 0°C)		1.333 22 x E -1		kilo pascal (kPa)

\*The becquerel (Bq) is the SI unit of radioactivity; 1 Bq = 1 event/s.

\*\*The Gray (Gy) is the SI unit of absorbed radiation.

# TABLE OF CONTENTS

Section	Page
CONVERSION TABLE.....	iii
FIGURES.....	v
TABLES.....	vi
1 INTRODUCTION.....	1
2 NEURAL NETWORK ALGORITHM.....	3
2.1 NEURAL NETWORK DESIGN.....	3
2.2 NEURAL NETWORK TRAINING.....	4
2.3 DATA VECTOR DEFINITION.....	5
2.4 NEURAL NETWORK CODE LISTING.....	9
3 ADVECTION ALGORITHM.....	31
3.1 PROGRESSIVE VECTOR ADVECTION.....	31
3.2 WIND VECTOR SMOOTHING.....	32
3.3 ALGORITHM LISTINGS.....	34
3.3.1 Progressive vector advection listing.....	34
3.3.2 Advection smoothing listing.....	40
4 PERSISTENCE ALGORITHM.....	45
5 EVOLUTION ALGORITHM.....	47
6 SKILL SCORE ALGORITHMS.....	52
7 REFERENCES.....	60

## FIGURES

Figure		Page
1-1	General structure of the code.....	1
2-1	Neural network configuration .....	3
3-1	In cases of significant curvature to the wind field, the progressive vector method (A) retains more accuracy than the linear extra-polation method (B).....	31
3-2	Cloud advection calculation using a 4 <sup>th</sup> order fit for the EMDA.....	35
3-3	Cloud advection results .....	36
5-1	Evolution data feed: (a) forecast cycle tested in the current model configuration, (b) example of another forecast cycle the model must eventually handle .....	51
6-1	Performance matrix.....	53
6-2	Skill scores .....	54
6-3	20/20 score .....	55
6-4	Brier score .....	56

## TABLES

Table		Page
2-1	Skilled scores for NN forecasts (cloud fraction).....	6
2-2	Final predictors.....	8
4-1	Persistence model data requirements.....	46
5-1	Evolution module predictors.....	48
5-2	25 top-ranked predictors for EASA data sets .....	50

# SECTION 1 INTRODUCTION

This report presents a description of the final algorithms included in the Worldwide Cloud Prediction Model (WCPM) developed by Pacific-Sierra Research Corporation. Existing code and algorithms are representative of development through feasibility demonstration on a regional basis. Development beyond a feasibility level was not possible due to a lack of data supplied by DSWA. Examples of algorithm performance and skill scores results are presented in Poehls, Crandall, O'Rourke and Heikes (1997).

The forecast code is designed around a unified NN with major weather inputs representing *advection* of clouds, *persistence* of clouds, and *evolution* of clouds along with several influence parameters. A pixel-by-pixel neural network (NN) algorithm is adopted as the generalized approach to cloud forecasting. The approach is based upon the assumption that a forecast is possible based solely upon the past, current and approaching clouds to a single pixel. The pixel-by-pixel implementation was chosen to minimize and simplify the data input into the NN. Each pixel is treated separately and is only loosely connected to surrounding pixels through the latitude and longitude inputs. No formal synoptic weather inputs are employed in this approach. The structure of the code is illustrated in Figure 1-1. This final form is somewhat different from

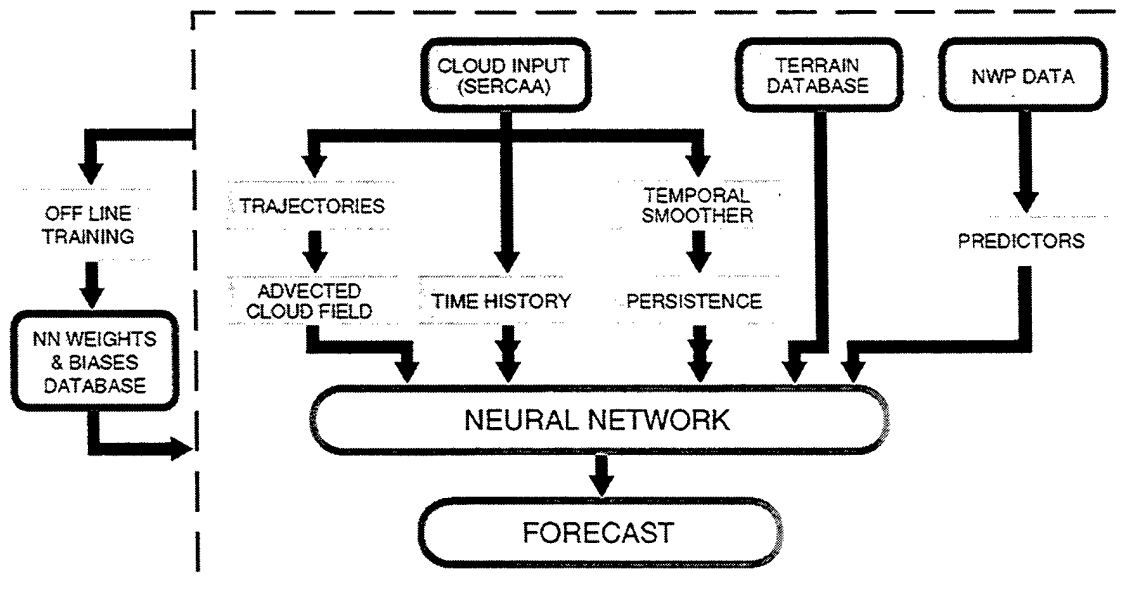


Figure 1-1. General structure of the code.

the original configuration that employed a separate NN for each module input and a NN to combine the individual forecasts. The latter was abandoned in favor of the unified approach to reduce the redundancy of the input parameters.

The weather inputs are divided into two categories: cloud observation data and meteorological parameter input. The *advection* and *persistence* modules represent the former while the *evolution* module represents the latter. For this study's purposes, the cloud observation data is taken from Support of Environmental Requirements for Cloud Analysis and Archive (SERCAA) level 3 nephanalysis. Navy Operational Global Atmospheric Prediction System (NOGAPS) numerical analysis and forecasts are used for the meteorological parameter inputs

The model will be described below in its final form, that is, merged into a single NN. The major pieces of the total process will be described in order of decreasing importance. The primary pieces are the NN itself and the advection algorithm which is pervasively utilized to identify and locate data in space and time. The NN is the dominant piece with all other algorithms directed toward providing either input or training data to the NN. The persistence and evolution algorithms are actually direct results of the advection process. One should therefore remember that although the algorithms are separately described, there was never any intention that they would perform as stand alone modules. Finally, although not directly associated with the forecast algorithm, the definition and calculation of skill scores will be discussed in Section 6.

## SECTION 2 NEURAL NETWORK ALGORITHM

The NN will be discussed in three parts. First, the general form of the NN is presented. Second, the training process is described. Third, the final selection of the input vectors was made based upon NN prediction performance. All are discussed in the following sections.

### 2.1 NEURAL NETWORK DESIGN.

The NN used in this project is the version of a feed-forward back-propagation (FFBP) originally written in C by Caudill (1994). Our version has been translated into FORTRAN 77. The NN is completely described in Caudill (1994). The FORTRAN code as we used it is listed in Section 2.4 of this document. This project did not progress beyond the use of the FFBP NN in its general form due to a severe lack of training data.

The fully-connected, feed-forward-back-propagation NN shown in Figure 2-1 was adopted for use on this project. The NN has 28 (the final number of inputs) input nodes, two hidden-layers

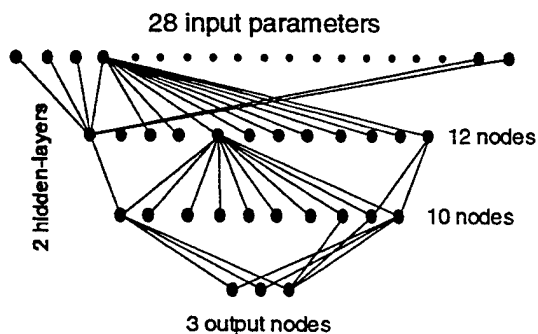


Figure 2-1. Neural network configuration.

(12 and 10 nodes each) and three output nodes for a total of 430 degrees of freedom. Several other variations on the number of hidden layers and the number of nodes in the hidden-layers were attempted. This was by no means an exhaustive study but several trends pointed toward the current selection. Greatly increasing the number of nodes in the hidden-layers significantly improved the training error but not the prediction error. A single hidden-layer performed more poorly. Reducing the hidden-layer nodes degraded the prediction capability.

## 2.2 NEURAL NETWORK TRAINING.

Training takes place on a batch of input vectors selected at random from the population of training vectors. The objective of training is to reduce the sum squared difference between the NN output and target cloud fields. The weight/bias set giving the least error is sought using a line minimization approach. Line minimization attempts to quickly hunt down the minimum of a two-dimensional curve by successively fitting parabolas to a region that brackets the minimum. This is usually more efficient than iterative methods where the minimum is found by taking a series of steps in the direction of greatest decreasing error (gradient descent), particularly if the minimum lies within a broad, shallow region of the curve. The error surface is actually multidimensional, the dimension depending on the number of weights and biases in the network. The search for a global minimum on the multidimensional error surface is reduced to a series two-dimensional searches by iteratively finding the minimum in first one direction, then another. Gradient descent moves in the direction of maximal error reduction. We employ a more efficient search that proceeds in the so-called conjugate gradient direction, which is a compromise between the previous search direction and that of gradient descent. The path defined by conjugate gradient directions tends to approach the minimum smoothly, eliminating inefficient zigzags inherent in the gradient descent approach.

The NN was extensively trained on the best and longest data set, the first six days of EMDA data (days 73 through 78). The following procedure was used:

1. An input file was created for all descriptors of each available (some were missing) hourly image.
2. All pixels were randomly selected from the first three days of data.
3. The NN was trained for 100 iterations on this training set.
4. The process was repeated for the second three days of data but the training was started with the previously obtained nodal weights.

The above procedure guaranteed that training included a distribution of available latitudes, longitudes, times of day and land types. (Dividing the data into two three-day pieces was based upon a computer limitation.) The NN was trained on a total of approximately 500,000 independent input vectors.

Training was stopped after 100 iterations in all cases. It was found that 95% of the training was accomplished in the first 25 to 35 iterations. Little improvement in training was realized after that point. In general, the training error varied from 15 to 20% when raw data was used as input; a 5% improvement was realized when median filtered data was used for training.

## 2.3 DATA VECTOR DEFINITION.

The final input vector definition was selected based upon an input parameter sensitivity study. The most straightforward method of determining which input parameters are important is to selectively omit parameters from the training process (Butler and Meredith, and Stogryn, 1996). The removal of a parameter can affect NN performance in three ways: 1) if the parameter is important, the NN performance is degraded, 2) if the parameter is unimportant, the NN performance is unchanged, and 3) if the parameter acts like a noise source, the NN performance is improved. Parameters that fall into the last category should be eliminated. Parameters that fall into the second category should be strongly considered for removal because their inclusion increases the training requirements and adds undesired degrees-of-freedom to the network.

A detailed study of all possible parameter combinations was obviously not performed. Instead, the study focused on the persistence input, the evolution parameters, and the influence parameters (latitude, longitude, land type, elevation, etc.). Table 2-1 presents the qualitative results of the study. Two important results emerge. First, the *elevation* input degrades the NN performance. Second, individually removing any of the many evolution parameters does not affect the NN performance, however, removing all of the evolution parameters degrades NN performance.

Based upon these results, the evolution parameters were re-evaluated in terms of the applicable atmospheric physics to select a much reduced input parameter set. The primary atmospheric condition that favors cloud formation is the uplift of warm moist air. This can be characterized by the NOGAPS *relative humidity*, *velocity divergence*, and *temperature* parameters at various altitudes. A new evolution predictor set of relative humidity, velocity divergence and temperature at five altitudes (Sea level, 100, 300, 500 and 850 hPa) was tested. Five altitudes provided redundant information. Two altitudes ( 850 and 500 mBars) provided the best compromise. Temperature was found to provide no meaningful NN performance and was eliminated from the predictors. The final predictors are listed in Table 2-2. The basic results reflect the most important predictors found by others. In reviewing the predictors (used and not used) it is important to remember that these were chosen based upon NN performance with a particular, limited set of tropical cloud data. Other scenarios might require some additions or adjustments to these predictors. More extensive NN training might reduce the training error and result in additional predictors becoming important.

Table 2-1. Skill scores for NN forecasts (cloud fraction).

Training Data	Sharp Obs.	Sharp For.	Brier	ESS	G20/20
<b>2 hour forecast</b>					
all*	0.97	0.67	0.12	0.26	0.62
elevation removed	0.97	0.77	0.13	0.33	0.67
lat/lon removed	0.97	0.77	0.14	0.21	0.67
longitude removed	0.97	0.70	0.13	0.32	0.62
land type removed	0.97	0.54	0.15	0.27	0.50
evol removed	0.97	0.71	0.11	0.32	0.65
evol removed except div850	0.97	0.74	0.12	0.32	0.67
elev. evolution <500 removed	0.97	0.74	0.12	0.29	0.66
div @ 850,500 only†	0.97	0.70	0.12	0.22	0.64
rh @ 850,500 only†	0.97	0.71	0.12	0.36	0.65
tmp @ 850,500 only†	0.97	0.74	0.11	0.39	0.67
temp & div @ 850,500 only†	0.97	0.75	0.12	0.33	0.67
rh & div @ 850,500 only†	0.97	0.73	0.12	0.39	0.66
tmp & rh @ 850,500 only†	0.97	0.76	0.12	0.32	0.68
evol @ 850,500 only†	0.97	0.68	0.12	0.29	0.63
<b>3 hour forecast</b>					
all*	0.97	0.67	0.13	0.28	0.60
elevation removed	0.97	0.75	0.13	0.31	0.66
lat/lon removed	0.97	0.78	0.14	0.19	0.67
longitude removed	0.97	0.68	0.13	0.32	0.61
land type removed	0.97	0.51	0.16	0.25	0.47
evol removed	0.97	0.69	0.12	0.32	0.63
evol removed except div850	0.97	0.71	0.12	0.30	0.64
elev. evolution <500 removed	0.97	0.71	0.12	0.31	0.64
div @ 850,500 only†	0.97	0.68	0.12	0.22	0.63
rh @ 850,500 only†	0.97	0.69	0.13	0.33	0.63
tmp @ 850,500 only†	0.97	0.73	0.12	0.31	0.66
temp & div @ 850,500 only†	0.97	0.74	0.12	0.33	0.66
rh & div @ 850,500 only†	0.97	0.71	0.13	0.33	0.64
tmp & rh @ 850,500 only†	0.97	0.75	0.12	0.34	0.67
evol @ 850,500 only†	0.97	0.66	0.12	0.30	0.61
<b>6 hour forecast</b>					
all*	0.97	0.64	0.13	0.30	0.58
elevation removed	0.97	0.75	0.14	0.31	0.66
lat/lon removed	0.97	0.74	0.14	0.17	0.64
longitude removed	0.97	0.68	0.14	0.29	0.60

Table 2-1. Skill scores for NN forecasts (cloud fraction) (Continued).

<b>6 hour forecast (continued)</b>					
land type removed	0.97	0.48	0.18	0.21	0.44
evol removed	0.97	0.61	0.13	0.32	0.57
evol removed except div850	0.97	0.68	0.13	0.28	0.63
elev. evolution <500 removed	0.97	0.67	0.12	0.30	0.61
div @ 850,500 only†	0.97	0.65	0.13	0.27	0.59
rh @ 850,500 only†	0.97	0.67	0.14	0.30	0.60
tmp @ 850,500 only†	0.97	0.73	0.13	0.30	0.66
temp & div @ 850,500 only†	0.97	0.73	0.13	0.26	0.65
rh & div @ 850,500 only†	0.97	0.70	0.14	0.30	0.63
tmp & rh @ 850,500 only†	0.97	0.74	0.13	0.26	0.66
evol @ 850,500 only†	0.97	0.66	0.12	0.19	0.61
<b>9 hour forecast</b>					
all*	0.97	0.59	0.13	0.37	0.55
elevation removed	0.97	0.74	0.13	0.31	0.66
lat/lon removed	0.97	0.77	0.14	0.26	0.66
longitude removed	0.97	0.72	0.14	0.29	0.62
land type removed	0.97	0.49	0.17	0.18	0.45
evol removed	0.97	0.59	0.14	0.27	0.54
evol removed except div850	0.97	0.73	0.13	0.32	0.66
elev. evolution <500 removed	0.97	0.65	0.12	0.38	0.59
div @ 850,500 only†	0.97	0.72	0.12	0.24	0.65
rh @ 850,500 only†	0.97	0.69	0.14	0.27	0.61
tmp @ 850,500 only†	0.97	0.78	0.13	0.33	0.68
temp & div @ 850,500 only†	0.97	0.71	0.13	0.22	0.63
rh & div @ 850,500 only†	0.97	0.71	0.13	0.32	0.64
tmp & rh @ 850,500 only†	0.97	0.73	0.13	0.33	0.65
evol @ 850,500 only†	0.97	0.70	0.12	0.32	0.64
<b>12 hour forecast</b>					
all*	0.97	0.62	0.13	0.28	0.56
elevation removed	0.97	0.72	0.13	0.33	0.65
lat/lon removed	0.97	0.81	0.15	0.17	0.67
longitude removed	0.97	0.75	0.14	0.27	0.64
land type removed	0.97	0.57	0.17	0.18	0.51
evol removed	0.97	0.62	0.13	0.25	0.56
evol removed except div850	0.97	0.81	0.13	0.23	0.71
elev. evolution <500 removed	0.97	0.67	0.12	0.22	0.60
div @ 850,500 only†	0.97	0.74	0.13	0.18	0.65
rh @ 850,500 only†	0.97	0.68	0.13	0.24	0.61
tmp @ 850,500 only†	0.97	0.81	0.13	0.28	0.71
temp & div @ 850,500 only†	0.97	0.72	0.13	0.32	0.64
rh & div @ 850,500 only†	0.97	0.73	0.13	0.30	0.65
tmp & rh @ 850,500 only†	0.97	0.76	0.13	0.30	0.67
evol @ 850,500 only†	0.97	0.76	0.13	0.26	0.68

\*This set has a duplicate t0 parameter included.

†These sets have elevation removed

Table 2-2 Final Predictors

NN Predictors
UT of forecast time
$\Delta t$ before forecast
Latitude
Longitude
Advection cloud fraction
Advection cloud top temperature
TCF at $t_0$
CTT at $t_0$
TCF at $t_0-1$ hour
CTT at $t_0-1$ hour
$\Delta t$ from forecast
TCF at $t_0-3$ hour
CTT at $t_0-3$ hour
$\Delta t$ from forecast
TCF at $t_0-6$ hour
CTT at $t_0-6$ hour
$\Delta t$ from forecast
TCF at $t_0-12$ hour
CTT at $t_0-12$ hour
$\Delta t$ from forecast
Clouds/no clouds flag
Relative humidity @ 850 hPa
Relative humidity @ 500 hPa
Velocity Divergence @ 850 hPa
Velocity Divergence @ 500 hPa
TCF at $t_0-24$ hours (Averaged over past 3 days)
CTT at $t_0-24$ hours (Averaged over past 3 days)
Land type

## 2.4 NEURAL NETWORK CODE LISTING.

A listing for the complete NN algorithm discussed in Section 2.1 is presented here for reference. This is a stand alone code that requires a previously calculated training weight set and properly formatted input data of the type described in Section 2.3. The codes are written in FORTRAN.

```
c*****
c
c Feed Forward Backpropagation (FFBP) Neural Network (NN)
c
c Routines:
c
c main                main control program
c do_forward_pass     propagate input activity forward thru network
c do_out_forward      do output layer, forward pass
c do_mid_forward      do middle layer, forward pass
c display_output      display output of network
c do_back_pass        propagate error activity backward thru network
c do_out_error        compute output layer errors
c do_mid_error        compute middle layer errors
c adjust_out_wts      adjust output layer weights
c adjust_mid_wts      adjust middle layer weights
c check_out_error     check to see if network knows all patterns yet
c initialize_net      do network initialization
c randomize_wts       randomize wts on middle & output layers
c read_data_file      read input/desired out patterns from data file
c display_mid_wts     output the weights on the middle layer neurodes
c display_out_wts     output the weights on the output layer neurodes
c
c*****

c-----
c MAIN PROGRAM
c-----
c
c   program main
c-----
c
c COMMON BLOCKS
c-----
c
c   include 'param.cmn'
c   include 'wts.cmn'
c   include 'wts_old.cmn'
c   include 'patts.cmn'
c   include 'errs.cmn'
c   include 'errs_old.cmn'
c   include 'nod_out.cmn'
c   include 'io.cmn'
c-----
```

c OPEN SETUP FILE, INITIALIZE VARIABLES, ETC.

c-----

```
lun_init = 1
lun_forefile = 2
lun_logfile = 3
lun_infile = 4
lun_wts_in = 5
lun_wts_out = 7
```

```
VECTORS_SEQUENTIAL = 1
IN_SIZE = 1
```

```
open(lun_init,file='nn_2mlc.ini',form='formatted')
```

```
read(lun_init,*) TRAIN_NETWORK
read(lun_init,*) PREDICTION
read(lun_init,*) IN_SIZE
read(lun_init,*) MID1_SIZE
read(lun_init,*) MID2_SIZE
read(lun_init,*) OUT_SIZE
read(lun_init,*) VECTORS_SEQUENTIAL
read(lun_init,*) VECTORS_RANDOMLY
read(lun_init,*) READ_WTS
read(lun_init,*) RANDOMIZE
read(lun_init,*) BETA
read(lun_init,*) BETA_UP
read(lun_init,*) BETA_UP2
read(lun_init,*) BETA_DN
read(lun_init,*) BETA_DN2
read(lun_init,*) ALPHA
read(lun_init,*) AL_UP
read(lun_init,*) AL_UP2
read(lun_init,*) AL_DN
read(lun_init,*) AL_DN2
read(lun_init,*) GAMMA
read(lun_init,*) STANDARD_ERR
read(lun_init,*) NUMSETS
write(*,*) 'Number of sets: ',NUMSETS
read(lun_init,*) MAX_ITERATIONS
```

```
read(lun_init,'(a14)') infile
read(lun_init,'(a14)') forefile
read(lun_init,'(a14)') logfile
read(lun_init,'(a14)') wts_in
read(lun_init,'(a14)') wts_out
```

```
close(lun_init)
```

c-----

c INITIALIZE MORE VARIABLES

c-----

```

T      = 1
F      = 0
ERR    = -1
MAXPATS = 100000
PRINT_ERRS = 0
PRINT_TO_OUTPUT= 0
VALMOD = 1.
LEARNED_ALL = F
STANDARD_ERR = OUT_SIZE*STANDARD_ERR
BETA_MAX = 1.0
ALPHA_MAX = 1.0

```

```

-----
c INITIALIZE NETWORK
-----

```

```

  call initialize_net()

```

```

-----
c SECTION FOR TRAINING NETWORK
-----

```

```

  IF (TRAIN_NETWORK .eq. T) THEN

```

```

    open(lun_logfile,file=logfile,form='formatted')

```

```

-----
c PUT PATTERNS INTO NN MAX_ITERATIONS TIMES
-----

```

```

  do 20 ir=1,MAX_ITERATIONS

```

```

    do 21 ip=1,numpats
      if (VECTORS_RANDOMLY .eq. T) ipatt = rand(0)*numpats + 1
      if (ipatt .gt. numpats) ipatt = numpats
      if (VECTORS_SEQUENTIAL.eq. T) ipatt = ip

```

```

      call do_forward_pass(ipatt)
      call do_back_pass(ipatt)
      patt_err_check = tot_out_error(ipatt)

```

```

      iteration_count = iteration_count + 1
21    continue

```

```

      do 22 ipatt=1,numpats
        call do_forward_pass(ipatt)
        call do_out_error(ipatt)
22    continue

```

```

      final_err_check_old = final_err_check
      call check_out_error()

```

```

      if (final_err_check .gt. final_err_check_old)BETA=BETA*BETA_DN
      if (final_err_check .lt. final_err_check_old)BETA=BETA+BETA_UP

```

```

if (BETA .gt. BETA_MAX) BETA = BETA_MAX

if (final_err_check .gt. final_err_check_old)ALPHA=ALPHA*AL_DN
if (final_err_check .lt. final_err_check_old)ALPHA=ALPHA+AL_UP
if (ALPHA .gt. ALPHA_MAX) ALPHA = ALPHA_MAX

err_percent = final_err_check/(numpats*OUT_SIZE)

if (err_percent .lt. 0.1) then
  BETA_DN = BETA_DN2
  BETA_UP = BETA_UP2
  AL_UP = AL_UP2
  AL_DN = AL_DN2
endif

write(*,100)ir,iteration_count,err_percent,BETA,ALPHA
100  format(1x,'Pass: ',i4,3x,'it: ',i9,3x,'Err: ',f6.4,3x,'Beta: ',
  $    f5.4,3x,'Alpha: ',f5.4)
write(*,*) '

if (final_err_check .lt. standard_err*numpats)learned_all=T
if (learned_all .eq. T) goto 99

20  continue
99  continue

c-----
c WRITE OUT FINAL NN WEIGHTS
c-----
write(*,*)'Posting final weights to file...'
open(lun_wts_out,file=wts_out,form='formatted')
call output_mid1_wts()
call output_mid2_wts()
call output_out_wts()
close(lun_wts_out)

c-----
c ALLOW OUTPUT NOW TO SEE HOW WELL NN IS DOING
c-----
PRINT_TO_OUTPUT = T

do 40 ipatt=1,numpats
  call do_forward_pass(ipatt)
  call do_out_error(ipatt)
40  continue

call check_out_error()

err_percent = final_err_check/(numpats*OUT_SIZE)

write(*,*) 'Final total error : ',err_percent

```

```

close(lun_logfile)

c-----
c END OF TRAINING SECTION CONTROL BLOCK
c-----
ENDIF

c-----
c NN ENGINE - PREDICTION SECTION
c-----
IF (PREDICTION .eq. T) THEN

    write(*,*)'Producing prediction ....'

    do 50 ipatt=1,numpats
        call do_forward_pass(ipatt)
        call do_out_error(ipatt)
50    continue

    call check_out_error()

    err_percent = final_err_check/(numpats*OUT_SIZE)

    write(*,*) 'Final error : ', err_percent

    open(lun_forefile,file=forefile,form='formatted')

    write(lun_forefile,*)((pred_pats(i,j),j=1,OUT_SIZE),i=1,numpats)
    write(lun_forefile,*)((pat_out(i,j), j=1,OUT_SIZE),i=1,numpats)

    close(lun_forefile)

    write(*,*)'Prediction complete ....'

c-----
c END OF PREDICTION SECTION CONTROL BLOCK
c-----
ENDIF

stop
end

c-----
c initialize_net()
c Do all the initialization stuff before beginning
c-----
subroutine initialize_net()

include 'param.cmn'
include 'io.cmn'

```

```

if (READ_WTS .eq. T) then
  open(lun_wts_in,file=wts_in,form='formatted')
  call read_mid1_wts()
  call read_mid2_wts()
  call read_out_wts()
  close(lun_wts_in)
endif

```

```

if (RANDOMIZE .eq. T) call randomize_wts()

```

```

call read_data_file()

```

```

iteration_count = 1

```

```

return
end

```

```

-----
c do_forward_pass(ipatt)
c control function for the forward pass through the network
-----

```

```

  subroutine do_forward_pass(ipatt)

  include 'param.cmn'

  call do_mid1_forward(ipatt) ! process forward pass, middle lyr 1
  call do_mid2_forward()    ! process forward pass, middle lyr 2
  call do_out_forward(ipatt) ! process forward pass, output lyr
  if (PRINT_TO_OUTPUT .eq. T) call display_output(ipatt)

  return
end

```

```

-----
c do_mid1_forward(ipatt)
c process the middle layer's forward pass
c The activation of middle layer's neurode is the weighted
c sum of the inputs from the input pattern, with sigmoid
c function applied to the inputs.
-----

```

```

  subroutine do_mid1_forward(ipatt)

  include 'param.cmn'
  include 'wts.cmn'
  include 'patts.cmn'
  include 'nod_out.cmn'

  real sum
  integer neurode, i

  do 10 neurode=1,MID1_SIZE

```

```

    sum = 0.0
    do 11 i=1,IN_SIZE ! COMPUTE WEIGHTED SUM OF INPUT SIGNALS
        sum = sum + mid1_wts(neurode,i)*pat_in(ipatt,i)
11    continue
        sum = 1./(1.+exp(-GAMMA*sum))
        mid1_out(neurode) = sum
10    continue

    return
    end

```

---

```

c do_mid2_forward()
c process the middle layer's forward pass
c The activation of middle layer's neurode is the weighted
c sum of the inputs from the input pattern, with sigmoid
c function applied to the inputs.

```

---

```

subroutine do_mid2_forward()

```

```

    include 'param.cmn'
    include 'wts.cmn'
    include 'patts.cmn'
    include 'nod_out.cmn'

```

```

    real sum
    integer neurode, i

```

```

    do 10 neurode=1,MID2_SIZE
        sum = 0.0
        do 11 i=1,MID1_SIZE ! COMPUTE WEIGHTED SUM OF INPUT SIGNALS
            sum = sum + mid2_wts(neurode,i)*mid1_out(i)
11    continue
            sum = 1./(1.+exp(-GAMMA*sum))
            mid2_out(neurode) = sum
10    continue

```

```

    return
    end

```

---

```

c do_out_forward()
c process the forward pass through the output layer
c The activation of the output layer is the weighted sum of
c the inputs (outputs from middle layer), modified by the
c sigmoid function.

```

---

```

subroutine do_out_forward(ipatt)

```

```

    include 'param.cmn'
    include 'wts.cmn'

```

```

include 'patts.cmn'
include 'nod_out.cmn'

real sum
integer neurode, i

do 10 neurode=1,OUT_SIZE
  sum = 0.0
  do 11 i=1,MID2_SIZE ! COMPUTE WEIGHTED SUM OF INPUT SIGNALS
    sum = sum + out_wts(neurode,i)*mid2_out(i)
11  continue
  sum = 1./(1.+exp(-sum))
  out_out(neurode) = sum
  pred_pats(ipatt,neurode) = sum
10  continue

return
end

C-----
c display_output(ipatt)
c Display the actual output vs. the desired output of the network.
c Once the training is complete, and the
c learned flag set to TRUE,
c then display_output sends its output to both the screen
c and to a text output file.
C-----
  subroutine display_output(ipatt)

include 'param.cmn'
include 'patts.cmn'
include 'nod_out.cmn'
include 'errs.cmn'
include 'io.cmn'

integer i

write(lun_logfile,*)'patt: ',ipatt
write(lun_logfile,*)'Desired Output:'
write(lun_logfile,100)(pat_out(ipatt,i),i=1,OUT_SIZE)
write(lun_logfile,*)'Actual Output:'
write(lun_logfile,100)(out_out(i),i=1,OUT_SIZE)
write(lun_logfile,*)'Error for pattern: ', tot_out_error(ipatt)

100 format(9(f7.5,1x))

return
end

C-----
c do_back_pass(ipatt)

```

c Process the backward propagation of error through network.

```
~~~~~  
subroutine do_back_pass(ipatt)
```

```
call do_out_error(ipatt)  
call do_mid2_error()  
call do_mid1_error()  
call adjust_out_wts()  
call adjust_mid2_wts()  
call adjust_mid1_wts(ipatt)
```

```
return  
end
```

```
~~~~~  
c do_out_error(ipatt)  
c Compute the error for the output layer neurodes, and current total  
c error.
```

```
~~~~~  
subroutine do_out_error(ipatt)
```

```
include 'param.cmn'  
include 'patts.cmn'  
include 'nod_out.cmn'  
include 'errs.cmn'
```

```
integer neurode  
real error_neurode,tot_error
```

```
tot_error = 0.0
```

```
do 10 neurode=1,OUT_SIZE  
  out_error(neurode) = pat_out(ipatt,neurode) - out_out(neurode)  
  error_neurode = abs(out_error(neurode))  
  tot_error = tot_error + error_neurode  
10 continue
```

```
tot_out_error(ipatt) = tot_error
```

```
return  
end
```

```
~~~~~  
c do_mid2_error()  
c Compute the error for the middle layer neurodes  
c This is based on the output errors computed above.  
c Note that the derivative of the sigmoid  $f(x)$  is  
c  $f'(x) = f(x)(1 - f(x))$   
c Recall that  $f(x)$  is merely the output of the middle  
c layer neurode on the forward pass.
```

*subroutine do\_mid2\_error()*

```
include 'param.cmn'  
include 'wts.cmn'  
include 'nod_out.cmn'  
include 'errs.cmn'
```

```
real sum  
integer neurode, i
```

```
do 10 neurode=1,MID2_SIZE  
  sum = 0.0  
  do 11 i=1,OUT_SIZE  
    sum = sum + out_wts(i,neurode)*out_error(i)  
11  continue
```

c APPLY THE DERIVATIVE OF THE SIGMOID HERE

```
  mid2_error(neurode)=mid2_out(neurode)*  
  $          (1.-mid2_out(neurode))*sum  
10 continue
```

```
  return  
  end
```

c-----  
c do\_mid1\_error()

c Compute the error for the middle layer neurodes  
c This is based on the output errors computed above.  
c Note that the derivative of the sigmoid  $f(x)$  is  
c  $f'(x) = f(x)(1 - f(x))$   
c Recall that  $f(x)$  is merely the output of the middle  
c layer neurode on the forward pass.

c-----

*subroutine do\_mid1\_error()*

```
include 'param.cmn'  
include 'wts.cmn'  
include 'nod_out.cmn'  
include 'errs.cmn'
```

```
real sum  
integer neurode, i
```

```
do 10 neurode=1,MID1_SIZE  
  sum = 0.0  
  do 11 i=1,MID2_SIZE  
    sum = sum + mid2_wts(i,neurode)*mid2_error(i)  
11  continue
```

c APPLY THE DERIVATIVE OF THE SIGMOID HERE

```

    mid1_error(neurode) = mid1_out(neurode)*
$      (1.-mid1_out(neurode))*sum
10 continue

    return
    end

```

```

C-----
c adjust_out_wts()
c Adjust the weights of the output layer. The error for the output
c layer has been previously propagated back to the middle layer.
c Use the Delta Rule with momentum term to adjust the weights.
C-----

```

*subroutine adjust\_out\_wts()*

```

include 'param.cmn'
include 'wts.cmn'
include 'wts_old.cmn'
include 'nod_out.cmn'
include 'errs.cmn'

```

```

integer weight, neurode
real learn,delta,alph

```

```

learn = BETA
alph = ALPHA

```

```

do 20 neurode=1,OUT_SIZE
  do 21 weight=1,MID2_SIZE
    delta =learn*out_error(neurode)*mid2_out(weight)
    out_wts(neurode,weight) = out_wts(neurode,weight) + delta +
$      out_wts_mom(neurode,weight)
    out_wts_mom(neurode,weight) = alph*(out_wts(neurode,weight) -
$      out_wts_old(neurode,weight))
    out_wts_old(neurode,weight) = out_wts(neurode,weight)
21 continue
20 continue

    return
    end

```

```

C-----
c adjust_mid2_wts()
c Adjust the middle layer weights using the previously computed errors.
c We use the Generalized Delta Rule with momentum term
C-----

```

*subroutine adjust\_mid2\_wts()*

```

include 'param.cmn'
include 'wts.cmn'

```

```

include 'wts_old.cmn'
include 'nod_out.cmn'
include 'errs.cmn'

integer weight, neurode
real learn,alph,delta

learn = BETA
alph = ALPHA

do 20 neurode=1,MID2_SIZE
do 21 weight=1,MID1_SIZE
delta = learn*mid2_error(neurode)*mid1_out(weight)
mid2_wts(neurode,weight) = mid2_wts(neurode,weight) + delta +
$ mid2_wts_mom(neurode,weight)
mid2_wts_mom(neurode,weight)=alph*(mid2_wts(neurode,weight)-
$ mid2_wts_old(neurode,weight))
mid2_wts_old(neurode,weight)=mid2_wts(neurode,weight)
21 continue
20 continue

return
end

```

```

c-----
c adjust_mid1_wts()
c Adjust the middle layer weights using the previously computed errors.
c We use the Generalized Delta Rule with momentum term
c-----

```

```

subroutine adjust_mid1_wts(ipatt)

```

```

include 'param.cmn'
include 'patts.cmn'
include 'wts.cmn'
include 'wts_old.cmn'
include 'nod_out.cmn'
include 'errs.cmn'

```

```

integer weight, neurode
real learn,alph,delta

```

```

learn = BETA
alph = ALPHA

```

```

do 20 neurode=1,MID1_SIZE
do 21 weight=1,IN_SIZE
delta = learn*mid1_error(neurode)*pat_in(ipatt,weight)
mid1_wts(neurode,weight) = mid1_wts(neurode,weight) + delta +
$ mid1_wts_mom(neurode,weight)
mid1_wts_mom(neurode,weight)=alph*(mid1_wts(neurode,weight)-
$ mid1_wts_old(neurode,weight))

```

```

        mid1_wts_old(neurode,weight)=mid1_wts(neurode,weight)
21  continue
20  continue

    return
    end

```

```

-----
c check_out_error()
c Check to see if the error in the output layer is below
c MARGIN*OUT_SIZE for all output patterns. If so, then assume the network
c has learned acceptably well. This is simply an arbitrary measure of how
c well the network has learned. Many other standards are possible.
-----

```

```

subroutine check_out_error()

```

```

    include 'param.cmn'
    include 'errs.cmn'

```

```

    integer i

```

```

    final_err_check = 0.0

```

```

    do 10 i=1,numpats

```

```

        final_err_check = final_err_check + tot_out_error(i)

```

```

10  continue

```

```

    return
    end

```

```

-----
c check_out_error_patt()
c Check to see if the error in the output layer is below
c MARGIN*OUT_SIZE for all output patterns. If so, then assume the network
c has learned acceptably well. This is simply an arbitrary measure of how
c well the network has learned many other standards are possible.
-----

```

```

subroutine check_out_error_patt(ipatt)

```

```

    include 'param.cmn'
    include 'errs.cmn'

```

```

    integer result

```

```

    result = T

```

```

    if (tot_out_error(ipatt) .ge. standard_err) result = F

```

```

    learned = result

```

```

    return
    end

```

```

c-----
c randomize_wts()
c Initialize the weights in the middle and output layers to
c random values between -0.25..+0.25
c-----

```

*subroutine randomize\_wts()*

```

include 'param.cmn'
include 'wts.cmn'
include 'wts_old.cmn'

integer neurode,i
real value

seed = 10000
value = rand(seed)

do 10 neurode=1,MID1_SIZE
do 11 i=1,IN_SIZE
value = rand(0) - 0.5
mid1_wts(neurode,i) = value*.8
mid1_wts_old(neurode,i) = value*.8
mid1_wts_mom(neurode,i) = 0.0
11 continue
10 continue

do 20 neurode=1,MID2_SIZE
do 21 i=1,MID1_SIZE
value = rand(0) - 0.5
mid2_wts(neurode,i) = value*.8
mid2_wts_old(neurode,i) = value*.8
mid2_wts_mom(neurode,i) = 0.0
21 continue
20 continue

do 30 neurode=1,OUT_SIZE
do 31 i=1,MID2_SIZE
value = rand(0) - 0.5
out_wts(neurode,i) = value*.8
out_wts_old(neurode,i) = value*.8
out_wts_mom(neurode,i) = 0.0
31 continue
30 continue

return
end

```

```

c-----
c read_data_file()
c Read in the input data file and store the patterns in pat_in

```

c and pat\_out.

c-----

*subroutine read\_data\_file()*

include 'param.cmn'  
include 'patts.cmn'  
include 'io.cmn'

integer youtsize,totsize  
integer ipatt  
integer tot  
integer iset  
integer numpats\_set

c

c NEW SECTION TO OBTAIN SELECTED PARAMETERS FROM INPUT VECTORS

c

integer no\_vect\_elem  
integer elem\_ids(47)  
integer out\_elem\_ids(47)  
real vect\_mask(47)  
real vect\_in(47)

num\_vect\_elem = IN\_SIZE  
num\_out\_elem = OUT\_SIZE

open(lun\_infile, file='vectmask.ini', form = 'formatted')  
do ielem = 1, num\_vect\_elem+num\_out\_elem  
  read(lun\_infile,\*) vect\_mask(ielem)  
enddo  
close(lun\_infile)

ielem\_cnt = 1  
do 9 ielem=1,num\_vect\_elem  
  if (vect\_mask(ielem) .eq. 1) then  
    elem\_ids(ielem\_cnt) = ielem  
    ielem\_cnt = ielem\_cnt + 1  
  endif

9 continue  
ielem\_cnt = ielem\_cnt - 1  
write(\*,\*) ielem\_cnt, ' input vector elements flagged for usage'

ioutelem\_cnt = 1  
do ioutelem=num\_vect\_elem+1,num\_vect\_elem+num\_out\_elem  
  if (vect\_mask(ioutelem) .eq. 1) then  
    out\_elem\_ids(ioutelem\_cnt) = ioutelem  
    ioutelem\_cnt = ioutelem\_cnt + 1  
  endif  
enddo  
ioutelem\_cnt = ioutelem\_cnt - 1

```

write(*,*) ioutelem_cnt, ' output elements flagged for usage'

c
c READ TRAINING OR FORECAST FILE
c

open(lun_infile, file=infile, form = 'formatted')

write(*,*) '

patt_cnt = 1

do 10 iset=1,NUMSETS
read(lun_infile,*)totsize,youtsize,numpats_set

write(*,*)'Input vector size : ',totsize
write(*,*)'Output vector size : ',youtsize
write(*,*)'Total set size : ',numpats_set

do 11 ipatt=1,numpats_set
read(lun_infile,*) (vect_in(tot),tot=1,totsize+youtsize)

do 111 ielem=1,ielem_cnt
pat_in(patt_cnt,ielem) = vect_in(elem_ids(ielem))
111 continue

do ioutelem=1,ioutelem_cnt
pat_out(patt_cnt,ioutelem) = vect_in(out_elem_ids(ioutelem))
enddo

patt_cnt = patt_cnt + 1
11 continue

10 continue ! END OF SET LOOP

totsize = ielem_cnt
numpats = patt_cnt - 1

write(*,*)'Total # vectors : ',numpats
write(*,*) '

close(lun_infile)

return
end

```

```

c-----
c display_mid1_wts()
c Display the weights on the middle layer neurodes
c-----
subroutine display_mid1_wts()

```

```

include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'

integer neurode, weight

write(lun_logfile,*)'Weights of Middle Layer neurodes: '

do 10 neurode=1,MID1_SIZE
  write(lun_logfile,*)'Mid Neurode # ',neurode
  do 11 weight=1,IN_SIZE
    write(lun_logfile,*) mid1_wts(neurode,weight)
11  continue
10  continue

return
end

```

```

-----
c display_mid2_wts()
c Display the weights on the middle layer neurodes
-----

```

```

  subroutine display_mid2_wts()

include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'

integer neurode, weight

write(lun_logfile,*)'Weights of Middle Layer 2 neurodes: '

do 10 neurode=1,MID2_SIZE
  write(lun_logfile,*)'Mid Neurode # ',neurode
  do 11 weight=1,MID1_SIZE
    write(lun_logfile,*) mid2_wts(neurode,weight)
11  continue
10  continue

return
end

```

```

-----
c display_out_wts()
c Display the weights on the middle layer neurodes
-----

```

```

  subroutine display_out_wts()

include 'param.cmn'
include 'wts.cmn'

```

```

include 'io.cmn'

integer neurode, weight

write(lun_logfile,*)'Weights of Output Layer neurodes: '

do 10 neurode=1,OUT_SIZE
  write(lun_logfile,*)'Mid Neurode # ',neurode
  do 11 weight=1,MID2_SIZE
    write(lun_logfile,*) out_wts(neurode,weight)
11  continue
10  continue

return
end

C-----
c output_mid1_wts()
C-----
  subroutine output_mid1_wts()

include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'

integer mid1_siz,in_siz

mid1_siz = MID1_SIZE
in_siz = IN_SIZE

write(lun_wts_out,*) mid1_siz
write(lun_wts_out,*) in_siz
write(lun_wts_out,*) mid1_wts

return
end

C-----
c output_mid2_wts()
C-----
  subroutine output_mid2_wts()

include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'

integer mid1_siz,mid2_siz

mid1_siz = MID1_SIZE
mid2_siz = MID2_SIZE

```

```
write(lun_wts_out,*) mid2_siz
write(lun_wts_out,*) mid1_siz
write(lun_wts_out,*) mid2_wts
```

```
return
end
```

```
C-----
c output_out_wts()
```

```
C-----
subroutine output_out_wts()
```

```
include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'
```

```
integer out_siz,mid2_siz
```

```
out_siz = OUT_SIZE
mid2_siz = MID2_SIZE
```

```
write(lun_wts_out,*) out_siz
write(lun_wts_out,*) mid2_siz
write(lun_wts_out,*) out_wts
```

```
return
end
```

```
C-----
c read_mid1_wts()
```

```
C-----
subroutine read_mid1_wts()
```

```
include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'
```

```
integer mid1_siz,in_siz
```

```
read(lun_wts_in,*) mid1_siz
read(lun_wts_in,*) in_siz
read(lun_wts_in,*) mid1_wts
```

```
return
end
```

```
C-----
c read_mid2_wts()
```

```
C-----
subroutine read_mid2_wts()
```

```

include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'

integer mid1_siz,mid2_siz

read(lun_wts_in,*) mid2_siz
read(lun_wts_in,*) mid1_siz
read(lun_wts_in,*) mid2_wts

```

```

return
end

```

```

C-----
c read_out_wts()
C-----

```

```

subroutine read_out_wts()

```

```

include 'param.cmn'
include 'wts.cmn'
include 'io.cmn'

integer out_siz,mid2_siz

read(lun_wts_in,*) out_siz
read(lun_wts_in,*) mid2_siz
read(lun_wts_in,*) out_wts

```

```

return
end

```

```

C errs.cmn

```

```

real mid1_error,mid2_error,out_error

common /errors/ mid1_error(80),mid2_error(80),out_error(32),
$ tot_out_error(100000),final_err_check,
$ final_err_check_old,patt_err_check,
$ patt_err_check_old

```

```

C io.cmn

```

```

integer lun_logfile
integer lun_forefile
integer lun_infile
integer lun_wts_in
integer lun_wts_out
character*20 logfile

```

```
character*20 forefile
character*20 infile
character*20 wts_in
character*20 wts_out
```

```
common /io/ lun_forefile,lun_infile,lun_wts_in,
$      lun_wts_out,lun_logfile,
$      logfile,forefile,outfile,infile,wts_in,wts_out
```

C *nod\_out.cmn*

```
real mid1_out,mid2_out,out_out
```

```
common /node_outputs/ mid1_out(80),mid2_out(80),out_out(32)
```

C *param.cmn*

```
integer T
integer F
integer ERR
integer MAXPATS
integer NUMSETS
integer IN_SIZE
integer MID1_SIZE
integer MID2_SIZE
integer OUT_SIZE
real MARGIN
integer MAX_ITERATIONS
integer MAX_PATT_ITERATIONS
real STANDARD_ER
integer VECTORS_SEQUENTIAL
integer VECTORS_RANDOMLY
integer EPOCH_TRAINING
integer READ_WTS
integer RANDOMIZE
integer PRINT_ERRS
```

```
integer iteration_count ! number of passes thru network so far
integer numpats        ! number of patterns in data file
integer learned        ! flag_if TRUE, network has a pattern
integer learned_all    ! flag_if TRUE, network has learned all patterns
real BETA
real ALPHA
real GAMMA
integer PRINT_TO_OUTPUT
real standard_err
integer ir
real valflt
integer valint
real valmod
real new_error
```

```
real old_error
integer patt_cnt
```

```
integer seed
```

```
common /parameters/ T,F,ERR,MAXPATS,NUMSETS,IN_SIZE,MID1_SIZE,
$ MID2_SIZE,OUT_SIZE,MARGIN,MAX_ITERATIONS,
$ MAX_PATT_ITERATIONS,STANDARD_ER,
$ VECTORS_SEQUENTIAL,VECTORS_RANDOMLY,
$ EPOCH_TRAINING,READ_WTS,RANDOMIZE,
$ PRINT_ERRS,iteration_count,numpats,
$ learned.BETA,ALPHA,GAMMA,PRINT_TO_OUTPUT,
$ standard_err,ir,valflt,valint,valmod,
$ learned_all,new_error,old_error,patt_cnt,
$ seed
```

### C *patts.cmn*

```
common /patterns/ pat_in(100000,200),
$ pat_out(100000,32),
$ pred_pats(100000,32)
```

### C *wts.cmn*

```
real mid1_wts,mid1_wts_mom
real mid2_wts,mid2_wts_mom
real out_wts ,out_wts_mom
```

```
common /weights/ mid1_wts(80,200),mid1_wts_mom(80,200),
$ mid2_wts(80, 80),mid2_wts_mom(80, 80),
$ out_wts (32, 80),out_wts_mom (32, 80)
```

### C *wts\_old.cmn*

```
real mid1_wts_old,mid2_wts_old,out_wts_old
```

```
common /weights_old/ mid1_wts_old(80,200),mid2_wts_old(80,80),
$ out_wts_old(32,80)
```

### C *errs\_old.cmn*

```
real mid1_error_old,mid2_error_old,out_error_old
```

```
common /errors_old/ mid1_error_old(80),mid2_error_old(80),
$ out_error_old(32)
```

## SECTION 3

### ADVECTION ALGORITHM

The cloud advection algorithm went through several incarnations before it was finalized. The earliest approaches were purposely simple:

- Wind vectors were estimated for the previous hour.
- Forecast time wind vectors were obtained by simply multiplying the 1 hour vectors by the forecast time.
- Clouds were moved based upon the vectors.

It was hoped that the NN would correct for poor wind estimates. Instead, it was found that poor wind estimates (when advection actually was the primary process) degraded the performance of the persistence and evolution inputs. Based upon this, the final advection algorithm contained two improvements: (1) a progressive wind vector advection algorithm replaced the simple single wind vector prediction, and (2) a smoothing algorithm was developed for the wind field.

#### 3.1 PROGRESSIVE VECTOR ADVECTION.

The previously employed advection algorithm was simple and efficient for short-term forecasts or wind fields with little curvature. When significant curvature exists, as occurs in flow about a major high or low pressure system, the simple linear approach produces extremely poor results. To rectify this a *progressive vector* advection module was created.

The clouds at a mesh point are advected using the following algorithm illustrated in Figure 3-1:

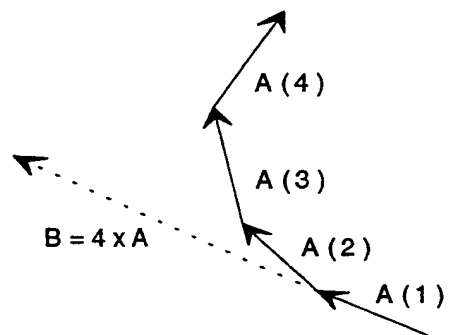


Figure 3-1. In cases of significant curvature to the wind field, the progressive vector method (A) retains more accuracy than the linear extra-polation method (B).

- The wind field for the most recent hour is assumed to be the best estimate of the wind field in the future.
- The clouds at a mesh point are advected forward 1 hour in time to a new mesh point using the wind vector at the current point.
- The wind vector at the new point is used to advect the clouds forward an additional 1 hour in time.
- The previous step is repeated until the desired forecast time is attained.

This procedure better retains the overall shape of the cloud formations as long as the current wind field accurately reflects the future wind field and the clouds are predominately advected (as opposed to evolved).

### 3.2 WIND VECTOR SMOOTHING.

The correlation analysis results in an inconsistent wind field, e.g. the field is not smooth and vectors often cross. To help alleviate (but not completely eliminate this problem) a smoothing process has been added to the wind field estimate. We have advection data defined on a 2D grid with lots of gaps – cloudless grid points with no good advection estimate. A weighted least squares smoother interpolator was developed.

The input data is on a grid of dimensions  $n_x \times n_y$ , with grid points at positions  $x = 1, 2, \dots, n_x$  and  $y = 1, 2, \dots, n_y$ . The input data consists of three pieces of data for each grid point:  $u(x, y)$  is the  $x$  component of the advection,  $v(x, y)$  is the  $y$  component, and  $w(x, y)$  is the weight.  $w$  is constructed from the correlation data: for good pixels,  $w$  is the correlation value (between 0 and 1 – no negative values); for bad (flagged) pixels,  $w$  is set to zero. For flagged pixels we should also set  $u$  and  $v$  to zero.

The data is fit by a set of smooth 2D basis functions. We'll specify the basis functions later, but for now let  $n_b$  be the number of basis functions used, and the basis functions are  $B_b(x, y)$  for  $b = 1, 2, \dots, n_b$ , defined for all  $x$  and  $y$ . The smoothed advection functions are linear superpositions of the basis functions, with some coefficients:

$$u_{smooth}(x, y) = \sum_{b=1}^{n_b} a_b B_b(x, y) \quad (3.1)$$

$$v_{smooth}(x, y) = \sum_{b=1}^{n_b} b_b B_b(x, y) \quad (3.2)$$

The coefficients are determined by doing a weighted fit to the advection data. This is the standard linear least squares fitting result, with weights. For the  $u$  data, define the variance

$$\sigma_x^2 \equiv \frac{1}{n_x n_y} \sum_{x=1}^{n_x} \sum_{y=1}^{n_y} w(x, y) [u(x, y) - u_{smooth}(x, y)]^2 . \quad (3.3)$$

Make the following definitions for the scalar  $UU$ , the vector  $BU$ , and the  $n_b \times n_b$  matrix  $BB$ :

$$UU \equiv \frac{1}{n_x n_y} \sum_{x, y} w(x, y) u(x, y)^2 . \quad (3.4)$$

$$BU_b \equiv \frac{1}{n_x n_y} \sum_{x, y} w(x, y) B_b(x, y) u(x, y) \quad (3.5)$$

$$BB_{bb'} \equiv \frac{1}{n_x n_y} \sum_{x, y} w(x, y) B_b(x, y) B_{b'}(x, y) \quad (3.6)$$

With these and some math, the variance is

$$\sigma_x^2 = UU - 2 \sum_b a_b BU_b + \sum_{b, b'} a_b a_{b'} BB_{b, b'} \quad (3.7)$$

Minimizing this with respect to  $a_b$  gives a solution in terms of the inverse of the matrix  $BB$ :

$$a_b = \sum_{b'} BB^{-1}_{b, b'} \cdot BU_{b'} , \quad (3.8)$$

and with this the variance is

$$\sigma_x^2 = UU - \sum_{b, b'} BU_b \cdot BB^{-1}_{b, b'} \cdot BU_{b'} . \quad (3.9)$$

The variance is useful to calculate, because it gives us a feeling for how well we're fitting the data.

If the basis functions were orthogonal, so that

$$BB_{bb'} = \frac{1}{n_x n_y} \sum_{x, y} w(x, y) B_b(x, y) B_{b'}(x, y) \quad (3.10)$$

was zero for  $b \neq b'$ , then the matrix would be diagonal and the inversion trivial. However, because of the arbitrary weights  $w$  in the equation, it is impossible to choose orthogonal basis functions. We will just choose simple basis functions, and have to live with the matrix inversion.

Figures 3-2 and 3-3 show an example calculation for the Mediterranean wind field. First, the north and east components of the wind field are estimated for individual cloudy pixels (Figures 3-2a and c). These are then smoothed and interpolated to produce the wind field used for advection (Figures 3-2b and d). The results of the advection are shown in Figure 3-3. Here, the original ( $T_0$ ) clouds are advected 12 hours based upon the old and the new smoothed  $T_0$  wind field. The results are compared to truth 12 hours later. Both approaches suffer from the fact that the cloud motion is not dominated by advection throughout the region; the clouds over southern Europe (to the left) are not moving but are evolving. Over northern Africa where advection is more dominant, the new model provides a better advection only forecast.

### 3.3 ALGORITHM LISTINGS.

Listing for the *progressive vector advection* discussed in Section 3.1 and the *wind vector smoothing* discussed in Section 3.2 are presented here for reference. These are algorithm codes and may require an appropriate driver for data input and output. The codes are written in IDL.

#### 3.3.1 Progressive Vector Advection Listing.

Two routines are listed here. The first, *ruv*, calculates the raw wind vectors and flags for a given set of successive one hour cloud images. The second, *correlat*, simply calculates the correlation coefficient between two images.

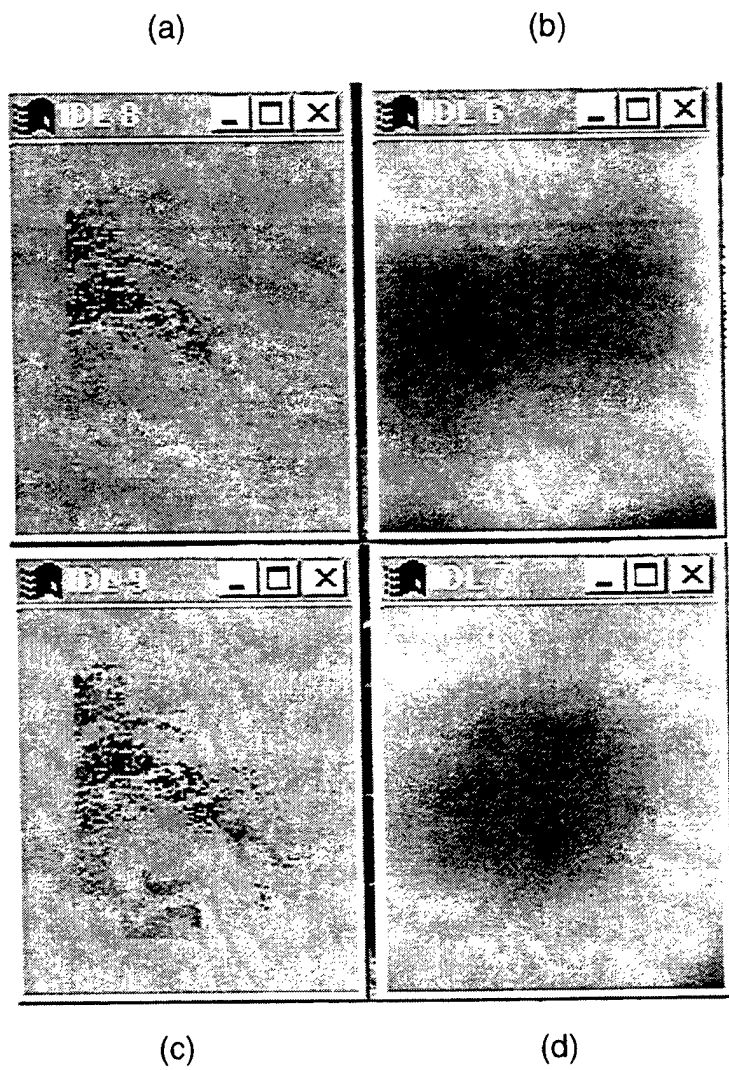


Figure 3-2. Cloud advection calculation using a 4<sup>th</sup> order fit for the EMDA.

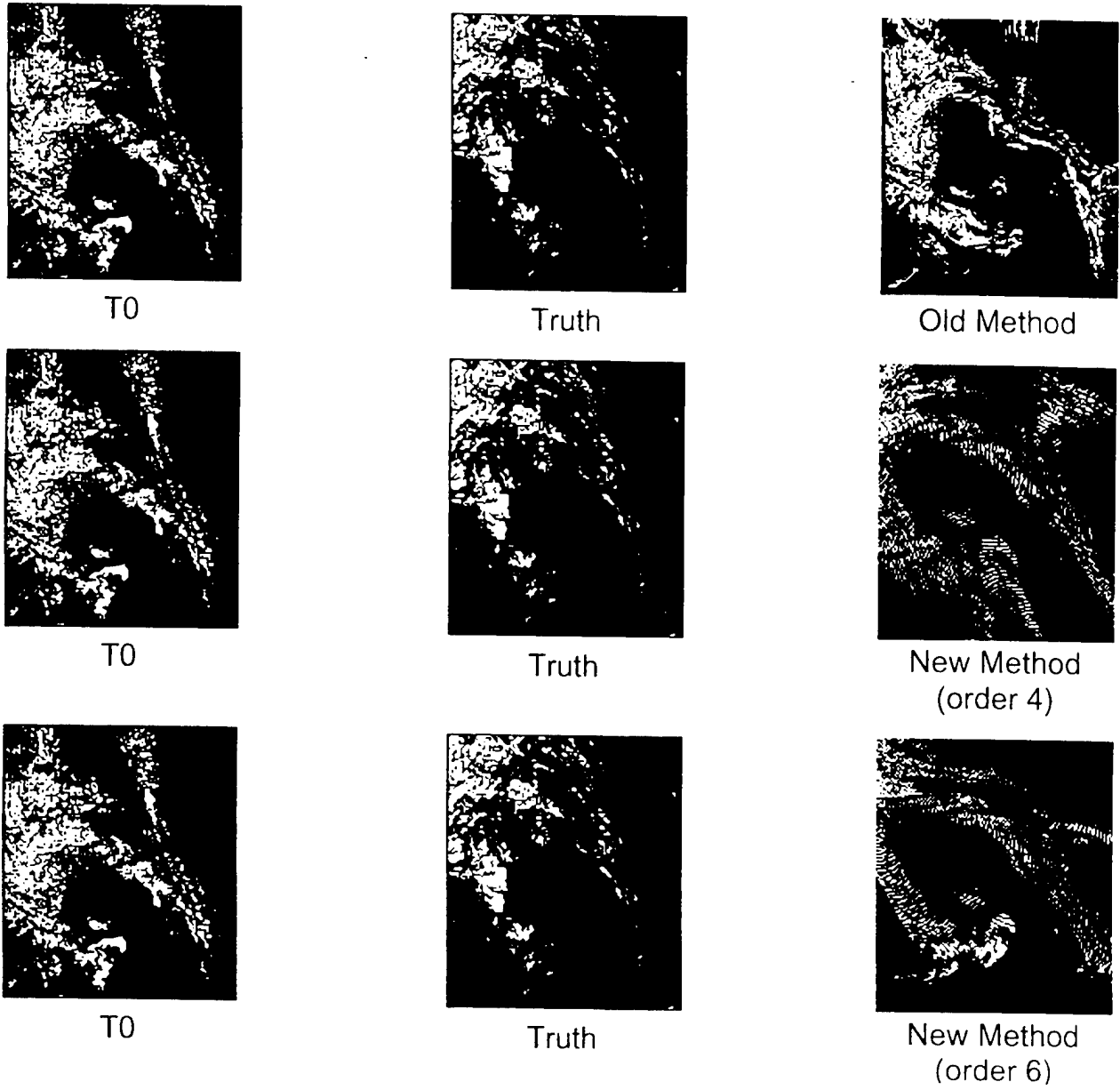


Figure 3-3. Cloud advection results.

The routine *ruv* is basically divided into two parts. The first part identifies those pixels that are eligible for vector estimation and flags those that are not. The second part uses a standard correlation process to calculate one hour wind vectors from the unflagged pixels.

```

.....
;routine rurv
; Calculate advection velocity
; Dimensions:
; Correlation window (current image)  $N_i \times N_i$  where  $N_i = 2n+1$ 
; Correlation test area (earlier image)  $N_m \times N_m$  where  $N_m = 4n+1$ 
; =  $2m+1$ 
.....
  rurvfilename = string(t0day,t0hour,format='(i2.2,"-",i2.2,".rurv")')
  flagfilename = string(t0day,t0hour,format='(i2.2,"-",i2.2,".flag")')
  irurvfile=0
  iflagfile=0
  junk = findfile(rurvfilename,count=irurvfile)
  junk = findfile(flagfilename,count=iflagfile)

  n = 15
  moffset = 7
  print, 'n=',n
  m = n + moffset
  ni = 2*n+1
  nm = 2*m+1
  no = 2*moffset+1
  f = bytarr(nm,nm)
  w = bytarr(ni,ni)
  r = fltarr(no,no)
  cv = fltarr(no,no)
  c0 = reform(c0,imag_x,imag_y)
  c1 = reform(c1,imag_x,imag_y)
  toofar = 5.

  advectflags = fltarr(imag_x,imag_y)

  ru = intarr(imag_x,imag_y)
  rv = intarr(imag_x,imag_y)

.....
; Calculation correlation offsets
.....

  if irurvfile eq 0 or iflagfile eq 0 then begin
    print,'Begining advection calculation'

    for xc = m,imag_x-1-m do begin
      for yc = m,imag_y-1-m do begin
        if c0(xc,yc) eq 0 then goto, j3

        w = c0(xc-n:xc+n,yc-n:yc+n)
        f = c1(xc-m:xc+m,yc-m:yc+m)
        for i = 0,2*moffset do begin
          for j = 0,2*moffset do begin

```

```

CORRELAT, w, f(i:i+2*n,j:j+2*n), cor, cov
r(i,j) = cor
cv(i,j) = cov
endfor
endfor
rmx = max(r,k)
ri = k mod no
rj = k/no
i = xc-moffset + ri      ; (i,j) is the point in c1 (earlier) that
j = yc-moffset + rj      ; best correlates with (xc,yc) in c0 (current)

```

```

; new as of 24 May 1996
advectflags(xc,yc) = rmx

```

```

; Check for bad points or unrealistic displacements

```

```

if cv(k) lt 500. then begin
  advectflags(xc,yc) = 0.
  goto, j3
endif
if rmx lt .3 then begin
  advectflags(xc,yc) = 0.
  goto, j3
endif
if abs(xc-i) gt toofar or abs(yc-j) gt toofar then begin
  advectflags(xc,yc) = 0.
  goto, j3
endif

```

```

; Calculate wind vectors

```

```

if advectflags(xc,yc) eq 0. then begin
  ru(xc,yc) = 0
  rv(xc,yc) = 0
endif else begin
  ru(xc,yc) = xc-i
  rv(xc,yc) = yc-j
endelse

```

```

j3:

```

```

  endfor
  print,FORMAT='(" .",,$)'
endfor
print,"
print,'End of correlation'

```

```

; Perform housekeeping

```

```

junk = where(advectflags ne 0,njunk)
print,'Number of non-zero weights: ',njunk
junk = where(c0 ne 0,njunk)

```

```

print,'Total cloudy pixels: ',njunk
junk = where(c0(m:imag_x-1-m,m:imag_y-1-m) ne 0,njunk)
print,'Cloudy pixels in correlation area: ',njunk

```

```

; Store the north-south and east-west vectors -- ru and rv -- and flags -- advectflags

```

```

openw, 2, rurvfilename
writeu, 2, ru
writeu, 2, rv
close,2

```

```

openw, 2, flagfilename
writeu, 2, advectflags
close,2

```

```

endif else begin

```

```

print,'Reading in previously calculated ru/rv and flags'

```

```

openr,1,rurvfilename
readu,1,ru
readu,1,rv
close,1

```

```

openr,1,flagfilename
readu,1,advectflags
close,1

```

```

endelse

```

```

PRO CORRELAT, X, Y, COR, COV

```

```

; Correlation and covariance subroutine

```

```

on_error,2 ; Return to caller if an error occurs.

```

```

; Means

```

```

nx = n_elements(x)
xmean = total(x) / nx
ymean = total(y) / nx

```

```

; Deviations

```

```

xx = x - xmean
yy = y - ymean

```

```

tt = total(xx*yy)
tx = total(xx^2)
ty = total(yy^2)

```

```

; Correlation

```

```

if tx eq 0 or ty eq 0 then cor = 0. else cor = tt / sqrt(tx*ty)

```

```

; Covariance

```

```

cov = tt / (nx-1)

```

```

return
end

```

### 3.3.2 Advection Smoothing Listing.

Two routines are listed here. The first, *vector\_smoothing* utilizes the ru, rv and flags output from Section 3.3.1 to calculate fitting coefficients for a smooth wind field according to the algorithm described in detail in Section 3.2. The second, *rurv\_smoothed*, calculates the complete, smoothed wind vector field given a set of smoothing coefficients.

```
; routine vector_smoothing
; dmc 28 May 1996
; new smoothing/interpolating rurv section
print,'Reading in previously calculated ru/rv and flags'
  openr,1,ruvfilename
  readu,1,ru
  readu,1,rv
  close,1

  openr,1,flagfilename
  readu,1,advectflags
  close,1
endelse

print,'Begin smoothing/interpolating of ru/rv'
nb = (np + 1) * (np + 2) / 2
ic = intarr(nb)
jc = ic
print,'Np: ',np
print,'Nb: ',nb

print,'Creating Ib and Jb'
b = 0
for i = 0, np do begin
  for j = 0, np - i do begin
    ic(b) = i
    jc(b) = j
    b = b + 1
  endfor
endfor
print,'Ib'
print,ic
print,'Jb'
print,jc

uu = 0.D
vv = 0.D
bu = dblarr(nb)
bv = bu
bb = dblarr(nb,nb)
```

```

badflag = .05
advectflags(0,0:imag_y-1) = badflag
advectflags(imag_x-1,0:imag_y-1) = badflag
advectflags(0:imag_x-1,0) = badflag
advectflags(0:imag_x-1,imag_y-1) = badflag

print,'Creating UU, VV, BU, BV, BB'

snx = dblarr(2*np+1)
sny = snx
snx(0) = np
sny(0) = np
for i = 1, 2*np do begin
  for j = 0, imag_x-1 do begin
    x = double(j)
    if j eq 0 then x = double(-50)
    if j eq imag_x-1 then x = double(j+50)
    snx(i) = snx(i) + (x/double(pixelscale))^i
  endfor
  for j = 0, imag_y-1 do begin
    y = double(j)
    if j eq 0 then y = double(-50)
    if j eq imag_y-1 then y = double(j+50)
    sny(i) = sny(i) + (y/double(pixelscale))^i
  endfor
endfor

for i = 0L, long(imag_x-1) do begin
  for j = 0L, long(imag_y-1) do begin

    if advectflags(i,j) gt 0. then begin
      x = double(i)
      if i eq 0 then x = double(-50)
      if i eq imag_x-1 then x = double(i+50)
      x = x/double(pixelscale)

      y = double(j)
      if j eq 0 then y = double(-50)
      if j eq imag_y-1 then y = double(j+50)
      y = y/double(pixelscale)

      uu = uu + advectflags(i,j) * double(ru(i,j))^2.
      vv = vv + advectflags(i,j) * double(rv(i,j))^2.
      for bi = 0, nb-1 do begin
        wxibyjb = double(advectflags(i,j)) * x^long(ic(bi)) * y^long(jc(bi))
        bu(bi) = bu(bi) + double(ru(i,j)) * wxibyjb
        bv(bi) = bv(bi) + double(rv(i,j)) * wxibyjb
      endfor
    endif
  endfor
endfor

```

```

mostflag = 0.
for i = 0L, long(imag_x-1) do begin
  for j = 0L, long(imag_y-1) do begin

    if advectflags(i,j) gt mostflag then begin
      x = double(i)
      if i eq 0 then x = double(-50)
      if i eq imag_x-1 then x = double(i+50)
      x = x/double(pixelscale)

      y = double(j)
      if j eq 0 then y = double(-50)
      if j eq imag_y-1 then y = double(j+50)
      y = y/double(pixelscale)

      af = advectflags(i,j) - mostflag
      for bi = 0, nb-1 do begin
        for bj = 0, bi do begin
          bb(bi,bj) = bb(bi,bj) + double(af) * x^long(ic(bi)+ic(bj)) *
            y^long(jc(bi)+jc(bj))
        endfor
      endfor
    endif
  endfor
endfor

for bi = 0, nb-1 do begin
  for bj = 0, bi do begin
    bb(bi,bj) = bb(bi,bj) + mostflag*snx(ic(bi)+ic(bj))*sny(jc(bi)+jc(bj))
  endfor
endfor

for bi = 0, nb-2 do begin
  for bj = bi+1, nb-1 do begin
    bb(bi,bj) = bb(bj,bi)
  endfor
endfor

nrml = total(advectflags)
uu = uu / double(nrml)
vv = vv / double(nrml)
bu = bu / double(nrml)
bv = bv / double(nrml)
bb = bb / double(nrml)

status = 0
print,'Inverting BB'
bbinv = invert(bb,status,double=1)
if status eq 0 then print,'Inversion successful'
if status eq 1 then begin

```

```

    print,'Inversion failed, singular matrix'
    retall
endif
if status eq 2 then print,'Inversion completed with loss of accuracy'

```

```

acoef = dblarr(nb)
bcoef = acoef
sigmax2 = uu
sigmay2 = vv

```

```

for i = 0, nb-1 do begin
  for j = 0, nb-1 do begin
    acoef(i) = acoef(i) + bbinv(i,j)*bu(j)
    bcoef(i) = bcoef(i) + bbinv(i,j)*bv(j)
    sigmax2 = sigmax2 - bu(i)*bbinv(i,j)*bu(j)
    sigmay2 = sigmay2 - bv(i)*bbinv(i,j)*bv(j)
  endfor
endfor
print,'X Var = ',sigmax2
print,'Y Var = ',sigmay2
print,'acoef = ',acoef
print,'bcoef = ',bcoef

```

```

; create a smoothed ru/rv for comparison purposes only

```

```

rus = dblarr(imag_x,imag_y)
rvs = rus
for i = 0,imag_x-1 do begin
  for j = 0,imag_y-1 do begin
    x = double(i)
    y = double(j)
    rus(i,j) = rurv_smoothed(acoef,x,y,ic,jc,double(pixelscale))
    rvs(i,j) = rurv_smoothed(bcoef,x,y,ic,jc,double(pixelscale))
  endfor
endfor

```

```

rurvsfilename = string(t0day,t0hour,format='(i2.2,"-",i2.2,".rurvs")')
openw, 2, rurvsfilename
writeu, 2, rus
writeu, 2, rvs
close,2
;

```

```

function rurv_smoothed, aa,xx,yy,iib,jjb,pixelscale

```

```

; dmc 28 May 1996
; return smoothed ru/rv

```

```

retval = 0.D
n = n_elements(aa)

```

```
sxx = xx / pixelscale
syy = yy / pixelscale
for i = 0, n-1 do begin
  retval = retval + aa(i)*sxx^iib(i)*syy^jjb(i)
endfor

return, retval
end
```

## SECTION 4

### PERSISTENCE ALGORITHM

Persistence is the tendency of weather to change slowly or to predictably repeat itself after some time interval. A forecast that merely persists current weather is usually the best short-term (0 to 3 hours) predictor. Some current tropical forecast models rely solely on simple persistence and a variation of it, diurnal persistence. Analyses by Salby, et al. (1991) indicate that a better persistence forecast might be obtained by including a more complete time history of cloud behavior. In particular, Salby, et al. noted strong regionally-dependent semi-diurnal and 4-day cycles associated with easterly waves in the tropics. A cloud history function that spans at least four days might improve forecasts.

The dominance of persistence in the SERCAA data areas is best represented by power spectral analysis. A complete description of the analysis is presented in Poehls, Crandall, O'Rourke and Heikes (1997). The results of the spectral analysis for EASA March 1993 tropical and mid-latitude ocean and land show a definite diurnal cycle over tropical land areas. No trends of any sort are apparent over ocean areas or at temperate latitudes. In fact, with the exception of the diurnal peaks, the spectra are representative of a white noise process with a very long term trend superimposed. The results for layers 3 and 4 represent pure white noise processes. These results do not preclude the presence of longer period cycles but more likely reflect poor resolution of the lower cloud layers by the SERCAA nephanalysis.

The proposed persistence modeling approach must be simplified based upon the above results. The proposed approach called for an auto-regressive model using a 6-day time series to capture the easterly wave 4-day cycle. The limited data supplied by DSWA clearly does not support such a model. Limited data also precludes model dependence upon geographic region and time of year. Given these constraints a simpler approach to a persistence model was adopted that only includes a 12 hour cloud history and an average diurnal input.

The 12 hour cloud history is simply input by including the current time cloud characterization along with a cloud characterization for 1, 3, 6, and 12 hours past. This data is meant to establish the near-time trend in cloud parameters.

The diurnal cycle in cloud parameters is input by averaging the cloud parameters from 24, 48, and 72 hours before the *forecast time*. This approach appears, and is, simple but was chosen for its robustness. The diurnal input can be averaged in several different ways and still be input. An adaptive recursive filter with a three day weight is an obvious choice for an operational system

but requires more data than was available to this analysis. The choice of weighting should be based upon information available upon longer term weather trends. This analysis used a simple three day average. A semi-diurnal or 4 day cycle can be input instead of the diurnal input.

Table 4-1 summarizes both the minimal and normal data requirements for the persistence algorithm. The minimum requirements refer to data requirements necessary for a cold start. Therefore the model can be started with only the previous day's data. Normal operation requires three previous days of data.

Table 4-1. Persistence model data requirements.

Minimum Requirements	Normal Requirements
$t_0$	$t_0$
$t_0 - 1$ (hours)	$t_0 - 1$ (hours)
$t_0 - 3$	$t_0 - 3$
$t_0 - 6$	$t_0 - 6$
$t_0 - 12$	$t_0 - 12$
$t_{\text{forecast}} - 24$	$t_{\text{forecast}} - \text{av} (24, 48, 72)$

Three quantities are input for each of the times (except diurnal) in Table 4-1. For each identified layer of clouds these include: (1) time delay from  $t_0$ ; (2) cloud fraction at the time delay; (3) cloud top temperature at the time delay.

## SECTION 5

### EVOLUTION ALGORITHM

Like persistence, the evolution algorithm depends on local characteristics such as topography, geography, latitude and time-of-day, but whereas the persistence and advection algorithms merely extrapolate cloud behavior in time and space, the evolution algorithm exploits atmospheric dynamics to predict clouds by engaging the output of a Numerical Weather Prediction (NWP) model. Since the military intends to consolidate all NWP functions under the Fleet Numerical Meteorological and Oceanography Center (FNMOC), and since NOGAPS is the Navy's global forecast model, it is likely that NOGAPS data will be the source of NWP data in future AF cloud forecast systems. Therefore, the decision was made to rely exclusively on NOGAPS as the source for NWP data.

Since NWP models generally do not predict clouds directly, it is necessary to relate the model output data to the cloud fields. The standard procedure for doing this is termed Model Output Statistics (MOS). The first step in the MOS approach is to define a set of *predictors* based on NWP forecast data. Predictors are not limited to NWP data and may include, for example, the current observed cloud fields. The predictors are then related to the forecast clouds (*predictands*) by means of a regression analysis on historical data. Our approach is similar except that we use a NN to relate predictors to predictands. The advantage of the NN approach is that possible nonlinear and cross-product relationships between predictors are automatically ferreted out by the NN to produce a better estimate of the predictand. The predictors are drawn from a pool of potential predictors that include elemental and derived variables based on NOGAPS data.

There is a large disparity in the resolutions of predictors based on NOGAPS data and predictands based on SERCAA data. NOGAPS provides a global analysis and a 12-hour forecast twice daily at 00 and 12 Z on a  $2.5 \times 2.5$  degree latitude/longitude grid. The resolution at  $60^\circ$  N is 139 km, decreasing to 278 km at the equator. In contrast, SERCAA data is available hourly (nominally) and the resolution of 16th-mesh SERCAA data at  $60^\circ$  N is 23.8 km, increasing toward the equator. The current NOGAPS operational model is higher resolution ( $0.75 \times 0.75$  degree) but unfortunately no archived data is available for the 1993 and 1994 times corresponding to the SERCAA data sets.

Table 5-1 shows the variables considered in the search for cloud field predictors. The first 6 variables are elemental NOGAPS model output data. The remaining variables, beginning with

divergence, are derived from the elemental variables. The height variable refers to the height of the pressure (hPa) surface. All variables, other than MSL pressure and surface (SFC) temperature, are defined on pressure surfaces listed across the top to the table. Vapor pressure (and thus relative humidity) is available only to 300 hPa. Divergence and vorticity are associated with vertical motion in the atmosphere at mid- to upper-latitudes and therefore likely to be correlated with clouds. Relative humidity is obviously linked with cloudiness. Temperature advection, vorticity advection, wind speed, and wind shear are often associated with developing storm systems. Temperature difference and thickness between pressure surfaces are measures of atmospheric stability.

Table 5-1. Evolution module predictors.\*

PREDICTOR	HEIGHT																	
	MSL	SFC	1000	850	700	500	400	300	200	150	100	50	70	20	30	10	925	
PRESSURE																		
HEIGHT																		
TEMPERATURE																		
VAPOR PRES																		
ZONAL VEL																		
MERIDNL VEL																		
DIVERGENCE																		
VORTICITY																		
REL HUMID																		
TEMP ADV																		
VORTICITY ADV																		
THICKNESS																		
WIND SPEED																		
WIND SHEAR																		
TEMP DIFF																		

- Blocked area indicate the heights for which predictor data is available.

Each predictor listed in Table 5-1 is used in three different ways. First, we simply take the predictor defined by the 12-hour forecast as it stands. Second, we subtract the zonal average from the 12-hour forecast value. Last, we define a trend based on the predictor at forecast time and its 12-hour forecast value. All calculations are performed on the NOGAPS  $2.5 \times 2.5$  degree grid and interpolated to the SERCAA 16th-mesh grid. Predictors are only compared to total cloud fraction and no attempt is made to discriminate predictors as a function of cloud layer, height, geography, or latitude zone. The 3 forms of 15 predictors at 17 heights result in pool of 618 potential predictors (not all variables are available at all heights).

A matrix correlation between predictor and predictand identified the predictors that showed the highest degree of association with the predictands. The best correlated predictors produced by this analysis significantly differed from those ranked high based on the contingency table. Visual comparisons of predictor and predictand in both cases led us to choose correlation as the best measure of association.

The correlation between predictor and predictand was then calculated for all times in each data set. The absolute values of correlation were averaged and ranked. Predictors that were related were eliminated to reduce redundancy. For example, if vapor pressure and relative humidity at a given height were both found to be highly correlated with total cloud fraction, then only the higher ranked predictor was kept. Similarly, only the higher ranked zonal wind or total wind speed was kept, since the zonal wind vector usually accounts for most of the wind speed magnitude. Also, only the higher ranked fundamental variable or its zonal perturbation was kept, not both. Table 5-2 shows the 25 top-ranked predictors for the March and July EASA data sets.

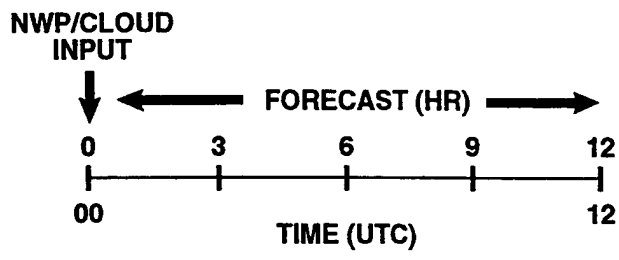
Once the best predictors were identified, a set of vectors was generated for NN training. Each training vector contains 37 input and 16 output elements. The input elements consists of predictors (25), current cloud fraction fields (4), elevation (1), time-of-day (2), latitude (1), longitude (2) and terrain slope (2). The output elements are 4 cloud fraction fields at 3, 6, 9, and 12 hours (16). The 25 top-ranked predictors were first calculated on the  $2.5 \times 2.5$  degree NOGAPS grid and then interpolated to the 16th-mesh SERCAA grid. Predictors were selected from 500 random locations within the region for each time in the data set. The times used for training are determined by the NWP forecast cycle. Only times where NWP data is available at the forecast time (Figure 5-1a) are used. The model has not been tested for times where NWP data is not synchronized with the forecast (Figure 5-1b). The last 12-hour period in the data set encompassing a NWP forecast cycle is reserved for validation. There are typically 15 times in each data set, excluding the last 12-hour period, where NWP data is synchronized with the

forecast time. As a result, the training set for each data set consists of about 7500 ( $500 \times 15$ ) training vectors.

Table 5-2. 25 top-ranked predictors for EASA data sets.

<b>MARCH</b>			<b>JULY</b>		
400 hPa	VAPP	TREND	400 hPa	VAPP	
700 hPa	VOR		500 hPa	RH	
850 hPa	VOR		300 hPa	VAPP	
500 hPa	SPEED		850 hPa	U_GRD	
300 hPa	SPEED		925 hPa	U_GRD	
700 hPa	SPEED		700 hPa	RH	
200 hPa	SPEED		1000 hPa	U_GRD	
400 hPa	SPEED		700 hPa	U_GRD	
700 hPa	SHEAR		500 hPa	U_GRD	
100 hPa	T_DIF		300 hPa	VAPP	
925 hPa	VOR		850 hPa	SHEAR	
150 hPa	SPEED		400 hPa	VAPP	TREND
100 hPa	SHEAR		200 hPa	DIV	
500 hPa	VAPP	TREND	400 hPa	T_DIF	TREND
50 hPa	T_DIF -		925 hPa	HGT -	
10 hPa	U_GRD -	TREND	850 hPa	HGT -	
700 hPa	RH	TREND	850 hPa	RH	
200 hPa	T_DIF -	TREND	400 hPa	U_GRD	
500 hPa	VOR		1000 hPa	HGT -	
300 hPa	THICK	TREND	0 MSL	PRES -	
300 hPa	TMP	TREND	10 hPa	U_GRD	TREND
1000 hPa	VOR		925 hPa	DIV -	
400 hPa	TMP	TREND	1000 hPa	DIV -	
850 hPa	VOR	TREND	700 hPa	HGT -	
850 hPa	HGT -		50 hPa	U_GRD -	

a.



b.

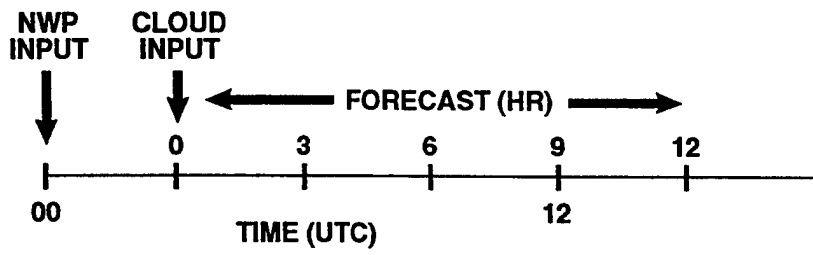


Figure 5-1. Evolution data feed: (a) forecast cycle tested in the current model configuration, (b) example of another forecast cycle the model must eventually handle.

## SECTION 6

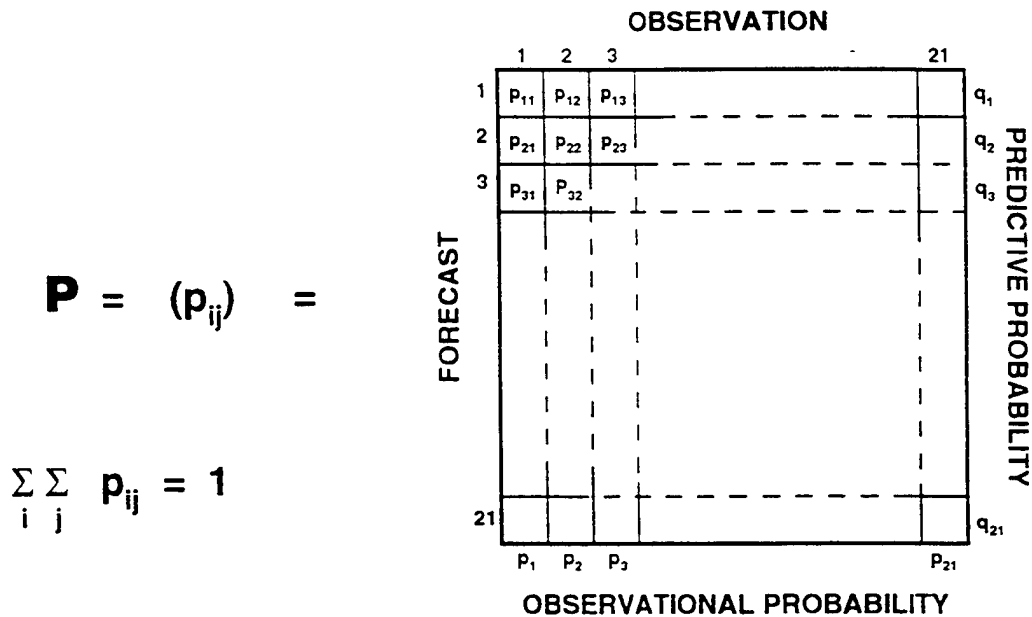
### SKILL SCORE ALGORITHMS

Skill scores provide a quantitative measure of model performance. Skill scores enable the comparison of forecast models based on alternate techniques and provide a means of measuring the effect of incremental improvements in the same model. The skill scores we have opted to use are the Equitable Skill Score (ESS), the 20/20 Score, and the Brier Score. We also look at the matrix correlation, global bias between forecast and observation, and forecast and observed sharpness. Sharpness is not strictly a performance statistic. It does not compare forecast to observational data. Rather it is a measure of the distribution of forecast or observed cloud field values taken individually.

The Equitable Skill Score (ESS), 20/20 Score, and Brier Score are all based on *performance matrices P* (Figure 6-1). A performance matrix is simply a normalized two-dimensional histogram of observed and forecast cloud field values. Each column  $j$  or row  $i$  represents a category of observation or forecast, respectively. For example, the columns might represent 5% increments in observed cloud fraction  $CF$ , with rows representing 5% increments in forecast  $CF$  as follows:

- Category 1: 0.00 CF < 0.05
- Category 2: 0.05 CF < 0.10
- Category 3: 0.10 CF < 0.15
- .
- .
- .
- Category 20: 0.90 CF < 0.95
- Category 21: 0.95 CF < 1.00

Each cell in the performance matrix contains the probability  $p_{ij} = n_{ij}/N$  that, given observation  $j$ , the forecast will be  $i$ . Here  $n_{ij}$  is the number of forecasts  $i$  for observation  $j$  and  $N$  is the total number of cases  $\sum n_{ij}$ .



**P** CONTINGENCY TABLE

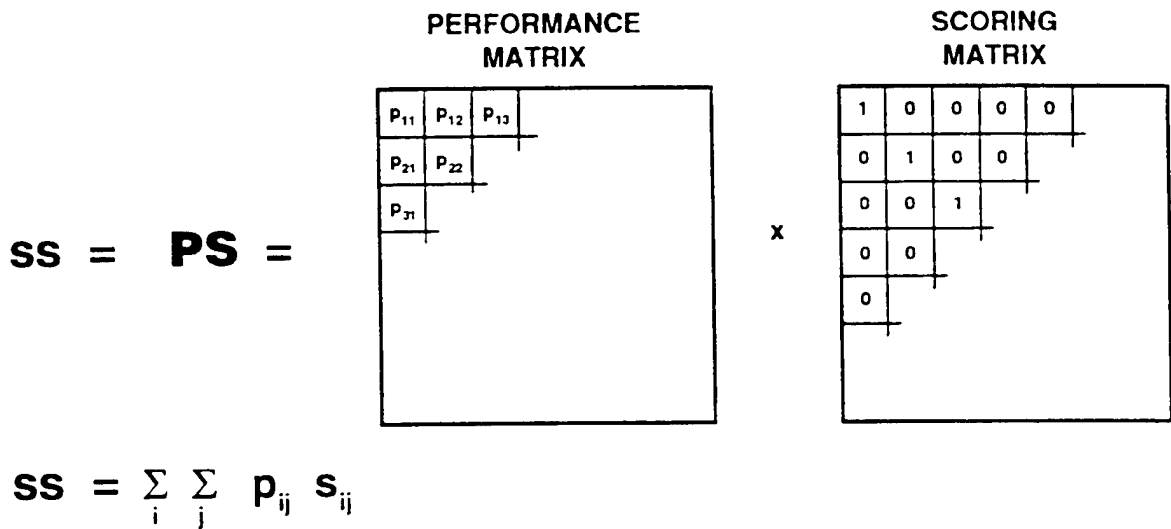
$p_{ij}$  RELATIVE FREQUENCY OF THE  $i$ th FORECAST FOR THE  $j$ th OBSERVATION

BIN	CLOUD FRACTION	CLOUD HEIGHT (m)
1	0.00 < 0.05	0 < 675
2	0.05 < 0.10	675 < 1350
⋮		
21	0.95 < 1.00	12,825 < 13,500

Figure 6-1. Performance matrix.

Skill score statistics are simply measures of the performance matrix probability distribution based on various *scoring matrices*  $S$ . A scoring matrix assigns a score to each cell in the performance matrix. An example of a scoring matrix is one that finds the relative frequency of correct forecasts (Figure 6-2). If the forecast is perfect, then all the entries in the performance matrix lie along on the diagonal where the forecast equals the observation. The scoring matrix shown in Figure 6-2 assigns a 1 to each correct forecast and 0 to all incorrect forecasts. Thus,  $PS = 1$  for a perfect forecast. The problem with this scoring matrix is that no credit is given for forecasts that are approximately correct (near, but not on the performance matrix diagonal).

- HOW GOOD IS THE FORECAST?
- ONE OBVIOUS MEASURE IS THE RELATIVE FREQUENCY OF CORRECT FORECASTS



- PERFECT FORECAST GIVES A SCORE OF 1

Figure 6-2. Skill scores.

One scoring matrix that credits nearly-correct forecasts is the 20/20 scoring matrix. The 20/20 score  $S_{20/20}$  measures the fraction of forecasts that are within  $\pm 20\%$  (i.e., within 4 categories) of the observed cloud field (Figure 6-3). The 20/20 scoring matrix  $S_{20/20}$  is given by

$$S_{20/20} = (s_{ij}) = 1 \quad \text{where } \max(1, j-4) \leq i \leq \min(21, j+4) \quad \text{and } j = 0, 1, \dots, 21 \quad (6.1)$$

## NUMBER OF FORECASTS WITHIN 20% OF OBSERVATIONS

**S** =  $(s_{ij}) =$

1	1	1	1	1	0	0
1	1	1	1	1	1	0
1	1	1	1	1	1	
1	1	1	1	1		
1	1	1	1			
0	1	1				
0	0					
0						

$$s_{ij} = 1 \quad \text{WHERE} \quad \max(1, j - 4) \leq i \leq \min(21, j + 4); \quad j = 0, 1, \dots, 21$$

FORECAST	SCORE*
PERFECT	1.00
RANDOM	0.38
AVERAGE	0.42

\*ASSUMING EQUALLY LIKELY OBSERVATIONS

Figure 6-3. 20/20 score.

The 20/20 score is 1 for a perfect forecast. To understand the significance of the value of 20/20 score  $S_{20/20}$  for an actual forecast, it is instructive to look at the 20/20 scores for random and constant forecasts. Consider a large number of equally likely observations. The probability of a particular observation falling in one of 21 possible performance categories is 1/21. For random forecasts, the probability of a forecast being in any one of 21 equally-sized categories is also 1/21. Therefore, the value of every cell in the performance matrix is  $p_{i,j} = 1/(21 \times 21)$  for a random forecast.

Now consider what happens if the forecast is always the same. Assume, for example, that a cloud fraction of 45 to 50% (category 10) is always forecast. Then, for equally likely observations  $i = 1, 2, \dots, 21$ , the forecast probability is

$$\begin{aligned}
 p_{ij} &= 1/21 & j = 10 \\
 p_{ij} &= 0 & j \neq 10.
 \end{aligned}
 \tag{6.2}$$

Applying the  $S_{20/20}$  scoring matrix to the random and constant performance matrices defined above, yields  $S_{20/20} = 0.38$  and  $0.42$ , respectively. Notice that the score is nonzero even for arbitrary forecasts.

The Brier score  $S_{Brier}$  is a measure of mean-squared error, so is particularly sensitive to off-diagonal forecasts (Figure 6-4). The Brier scoring matrix  $S_{Brier}$  is defined

### MEAN SQUARED DIFFERENCE BETWEEN FORECAST AND OBSERVATION

$$\mathbf{S} = (s_{ij}) =$$

.0000	.0025	.0010	1.0000
.0025	.0000		
.0010			
1.0000			.0000

$$s_{ij} = (F_i - O_j)^2 \text{ WHERE } F_i = \text{FORECAST, } O_j = \text{OBSERVED}$$

FORECAST	SCORE*
PERFECT	0.00
RANDOM	0.18
AVERAGE	0.20

\*ASSUMING EQUALLY LIKELY OBSERVATIONS

Figure 6-4. Brier score.

$$SBrier = (s_{i,j}) = (0.05)^2 (i - j)^2 \quad (6.3)$$

The Brier score for a perfect forecast is 0. Assuming equally likely observations, the Brier score for random and constant performance matrices are 0.18 and 0.20, respectively. Again, the score for an arbitrary forecast is not the extreme error value (the extreme being 1).

As noted above, the 20/20 and Brier scores have the undesirable characteristic that constant and random forecasts can be credited with significant forecast skill. Moreover, these scoring matrices are inequitable in the sense that, in cases where not all observations are equally likely, constant forecasts of some events lead to better scores than constant forecasts of other events. It is therefore desirable to devise a scoring matrix with the properties that (i) scores assigned to uncommon events, in terms of climatological probability, increase as climatological probability decreases and (ii) scores of zero are assigned to random and constant forecasts.

An Equitable Skill Score (ESS) matrix has been formulated by Gandin and Murphy (1992) and Gerrity (1992). A climatological probability vector can be defined from the performance matrix as the probability of occurrence of the  $j$ th observation

$$p = (p_j) = \sum_i p_{ij} \quad (6.4)$$

Similarly, a predictive probability vector can be defined as the probability of occurrence of the  $i$ th forecast

$$q = (q_i) = \sum_j p_{ij} \quad (6.5)$$

Now define

$$D_n = \frac{1 - \sum_{r=1}^n p_r}{\sum_{r=1}^n p_r} \quad (6.6)$$

$$R_n = \frac{1}{D_n} \quad (6.7)$$

$R_n$  is the ratio of the probability that an observation falls in a category greater than  $n$  to the probability that it falls into a category less than  $n$ . Following Gerrity, the ESS scoring matrix  $S_{ESS} = (s_{ij})$  is constructed as follows

$$s_{n,n} = K \left[ \sum_{r=1}^{n-1} R_r + \sum_{r=n}^{K-1} D_r \right] \quad n = 1, 2, \dots, K . \quad (6.8)$$

$$s_{m,n} = K \left[ \sum_{r=1}^{m-1} R_r + \sum_{r=m}^{n-1} (-1) + \sum_{r=n}^{K-1} D_r \right] \quad 1 \leq m < K, \quad m < n \leq K \quad (6.9)$$

$$s_{n,m} = s_{m,n} \quad 2 \leq n \leq K, \quad 1 \leq m < n \quad (6.10)$$

$$K = \frac{1}{K-1} . \quad (6.11)$$

$S_{ESS}$  has the desirable properties that, when multiplied by the performance matrix, perfect forecasts score 1, and random and constant forecasts score 0.

Another forecast skill diagnostic is sharpness. Sharpness is not a skill score but a measure of the individual cloud cover distribution of observed and forecast clouds. It measures the relative frequency of cases occupying the extreme categories of 0- to 20% and 80 to 100% cloud fraction. Observed and forecast sharpness are

$$S_O = \sum_{i=1}^5 p_i + \sum_{i=17}^{21} p_i \quad (6.12)$$

$$S_F = \sum_{i=1}^5 q_i + \sum_{i=17}^{21} q_i \quad (6.13)$$

Individual sharpness values have limited diagnostic utility. Only the relative values of observed and forecast sharpness have meaning. Most cloud forecast techniques tend to forecast mid-range cloud amounts. Comparing observed and forecast sharpness indicates whether the forecast model captures outlying cloud distributions, or whether it simply forecasts mid-range values. On the other hand, sharpness values can be misleading. For example, the sharpness for an observed 100% overcast and that for a 100% clear forecast are identical.

The last two forecast diagnostics are bias and correlation. Bias is simply the difference between observed and forecast values

$$B = \sum_{i=1}^{21} \sum_{j=1}^{21} (p_{ij} - q_{ij}) \quad . \quad (6.14)$$

Bias is zero for a perfect forecast. The matrix correlation  $C$  between forecast and observation is

$$C = \frac{\sum_{i=1}^N (F_i - \bar{F})(O_i - \bar{O})}{\sqrt{\sum_{i=1}^N (F_i - \bar{F})^2 \sum_{i=1}^N (O_i - \bar{O})^2}} \quad . \quad (6.15)$$

where  $F_i$  and  $O_i$  are the forecast and observation cloud field values at  $N$  image pixels, respectively. The overbar indicates the mean values of these quantities. Correlation  $C$  is one for perfect forecast.

## SECTION 7

### REFERENCES

Butler, C. T., and R. v. Z. Meredith, and A. P. Stogryn, "Retrieving Atmospheric Temperature Parameters from DMSP SSM/T-1 Data with a Neural Network," *J. Geophys. Res.*, Vol. 101, 1996, pp. 7075-7-83. (UNCLASSIFIED)

Caudill, M., *Neural Networks Primer*, (3<sup>rd</sup> Ed., Rev.), AI Expert, Miller Freeman Publication, 1994. (UNCLASSIFIED)

Gandin, L. S., and A. H. Murphy, "Equitable Skill Scores for Categorical Forecasts," *Mon. Weather Rev.*, Vol. 120, 1992, pp. 361-370. (UNCLASSIFIED)

Gerrity, J. P., "A Note on Gandin and Murphy's Equitable Skill Score," *Mon. Weather Rev.*, Vol. 120, 1992, pp. 2709-2712. (UNCLASSIFIED)

Poehls, K. A., D. M. Crandall, K. O'Rourke, and K. E. Heikes., *Worldwide Cloud Forecasts with Neural Networks*, Report 2692, Pacific-Sierra Research Corporation, Santa Monica, CA, 1977. (UNCLASSIFIED)

Salby, M. L., et al., "Analysis of Global Cloud Imagery from Multiple Satellites," *Bull. Amer. Meteor. Soc.*, Vol. 72, No. 4, 1991, pp. 467-480. (UNCLASSIFIED)

**DISTRIBUTION LIST  
DSWA TR-97-82**

**DEPARTMENT OF DEFENSE**

DEFENSE TECHNICAL INFORMATION  
2 CY ATTN: DTIC/OCF

DEFENSE THREAT REDUCTION AGENCY  
ATTN: SWET, DAVID MYERS  
2 CY ATTN: SWET, MAJ WELLS  
2 CY ATTN: SWI  
ATTN: WEL, L WITWER  
ATTN: WEL, MAJ T SMITH

AODTRA  
ATTN: CPX, LTC D. R. LITTLE  
ATTN: FCT-S, G BALADI  
ATTN: SWP

JOINT CHIEFS OF STAFF  
ATTN: J8 WAR FIGHTING DIV

**DEPARTMENT OF THE ARMY**

DEPUTY CHIEF OF STAFF FOR OPERATIONS  
AND PLANS  
ATTN: DAMO-NCZ

US ARMY RESEARCH LAB  
ATTN: ;SLCBR-S, TECH LIB

US ARMY RESEARCH LABORATORIES  
ATTN: AMSRL-SL-CE  
ATTN: J MARTIN  
ATTN: r CIONCO

WEST DESERT TEST CENTER  
ATTN: CHRIS BILTOFT  
ATTN: JIM BOWERS

**DEPARTMENT OF THE NAVY**

DEPUTY CHIEF OF NAVAL OPERATIONS  
ATTN: N514 BRANCH HEAD

NAVAL RESEARCH LABORATORY  
ATTN: CODE 5227, RESEARCH REPORT  
ATTN: SIMON CHANG

NAVAL SURFACE WARFARE CENTER  
ATTN: BSI, T. BAUER

**DEPARTMENT OF THE AIR FORCE**

AF WEATHER TECHNICAL LIBRARY  
ATTN: KAY MARSHALL

AIR FORCE SPACE COMMAND  
ATTN: LTOL CROSS

AIR FORCE WEATHER AGENCY/DNXM  
ATTN: MAJ RANDY LEFEVRE

AIR UNIVERSITY LIBRARY  
ATTN: AUL-LSE

HQ USAF/XOWX  
ATTN: XOWX

**DEPARTMENT OF ENERGY**

LAWRENCE LIVERMORE NATIONAL LAB  
ATTN: ALLEN JUHL  
ATTN: L-81, R PERRETT

LOS ALAMOS NATIONAL LABORATORY  
ATTN: A S MASON  
ATTN: J NORMAN  
ATTN: ESS-5, R W WHITAKER  
ATTN: WX-1, B SHAFER

**DEPARTMENT OF DEFENSE CONTRACTORS**

APPLIED RESEARCH ASSOCIATES, INC.  
ATTN: C NEEDHAM

ITT INDUSTRIES  
ATTN: DASIAC  
ATTN: DASIAC/DARE

LOGICN RDA  
RAY POPE  
ATTN: TOM MAXOLLA

MISSION RESEARCH CROP  
ATTN: BRUCE BAUER

PACIFIC-SIERRA RESEARCH CROP.  
2 CY ATTN: D CRANDALL  
ATTN: H BRODE

DSWA-TR- 97-82 (DL CONTINUED)

2CY ATTN: K HEIKES  
2 CY ATTN: K O\*ROUKE  
2 CY ATTN: K POEHLS

SCIENCE APPLICATION INTL CORP  
ATTN: T JARRETT

SCIENCE APPLICATION INTL CORP  
ATTN: J MANSHIP

SCIENCE APPLICATION INTL CORP  
ATTN: D BACON  
ATTN: J COCKAYNE  
ATTN: J JMC GAAGHAN

SCIENCE APPLICATION INTL CORP  
ATTN: J SONTOWSKI

THE AEROSPACE CORP  
ATTN: DR MIKE PLONSKI

THE TITAN CORPORATION  
ATTN: IAN SYKES

TITAN CORPORATION (THE)  
ATTN: R ENGLAND

TRW S. I. G.  
NORMAN LIPNER

WISIDYNE, INC.  
J DEVORE