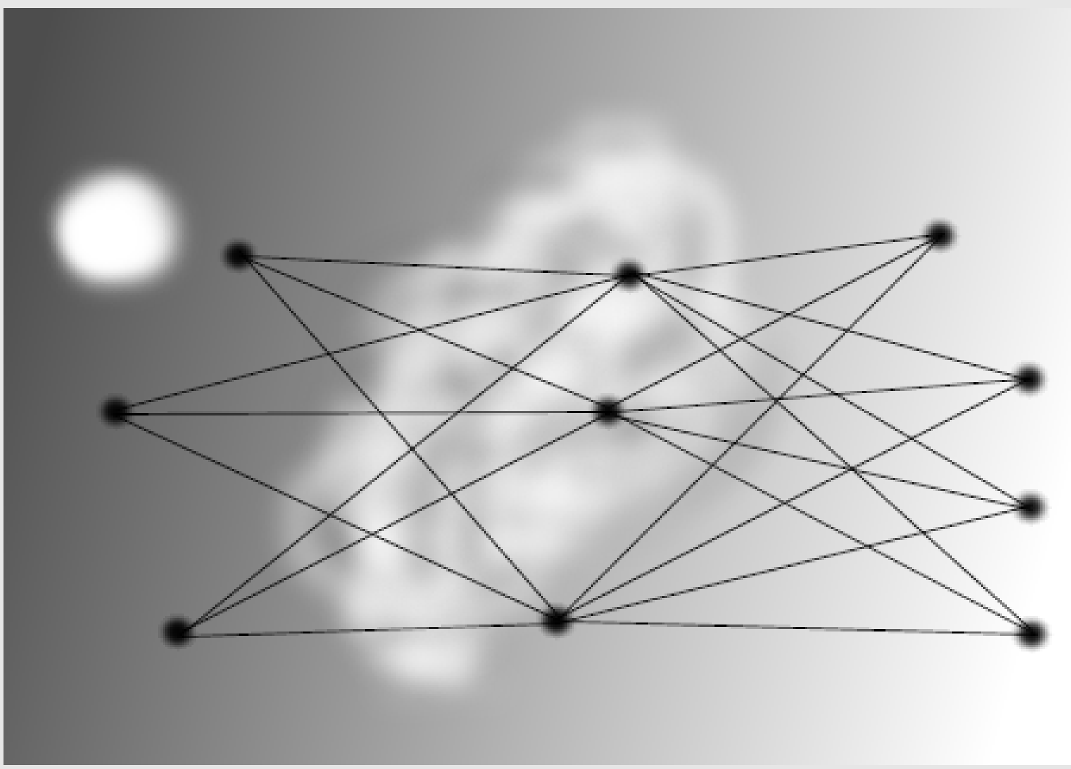
# *FluxNet*

**Version 4.0**

# User's Guide



# *FluxNet*

# User's Guide

Jeffrey R. Key

NOAA/NESDIS
Madison, Wisconsin

Axel J. Schweiger

Polar Science Center
Applied Physics Laboratory
University of Washington

June 1, 2001

CIMSS

## TABLE OF CONTENTS

# 1 INTRODUCTION

***FluxNet*** is a neural network version of *Streamer*, a radiative transfer model. Given a set of inputs consisting of surface, cloud, and atmospheric parameters, *FluxNet* will calculate downwelling and upwelling shortwave and longwave fluxes at the surface and the top of the atmosphere. The code is very small and extremely fast. Two versions of the network are currently available: one requiring temperature and humidity profiles in the input stream and one that uses total column water vapor rather than the profiles. Both were developed for global conditions.

This manual will deal only with information needed to run and modify *FluxNet*. For details about the physical basis of the model, the reader should refer to the *Streamer User's Guide*. Users who wish to modify or customize *FluxNet* should have some practical knowledge of neural networks, Fortran, C, and Unix programming.

## DIFFERENCES BETWEEN *STREAMER* AND *FLUXNET*

*FluxNet* is a backpropagation network. Like many Artificial Neural Networks (ANN), it becomes "hardwired," or inflexible, once it has been trained. Because of some of the limitations inherent in the neural network, and because of the time involved in creating the large data set needed to train the network, *FluxNet* is a much simpler model than *Streamer*. Its principle advantage is that it is faster than *Streamer* by two to four orders of magnitude (100 to 10,000 times), making it ideal for large jobs like image processing, which consist of thousands to millions of difference cases. Users of *Streamer* should be aware of the following limitations in *FluxNet*:

- Each "scene" (for example, a pixel in an image) can consist of just one surface type, open sea water, snow/ice, or vegetation, and one cloud layer.
- The shortwave bands (bands 106 – 129 in *Streamer*) and longwave bands (bands 1 – 105) have been consolidated into one shortwave and one longwave band; i.e., broadband calculations are done.
- The output file format is fixed, although it would be easy to modify the C or IDL source code.

There are also problems and peculiarities associated with neural networks:

- An artificial neural network is created by "training" it on a particular set of input data; it will behave unpredictably on inputs that fall outside the range of data provided in this training set. See *Input and Output* for the recommended range of values for each input parameter.
- If the values for certain parameters are unknown, a default value, rather than "0", should be used instead. See *Input and Output* for a list of recommended default values for each input parameter.

## ACKNOWLEDGMENTS

*FluxNet* is a neural network approximation of *Streamer*. Please refer to the *Streamer User's Guide* for more information regarding *Streamer* and its history. The *Stuttgart Neural Network Simulator* by Andreas Zell et al. (http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html) was used to train the neural network component of *FluxNet* and to convert it to C. *FluxNet* was designed and implemented by J. Key, E. Amano, J. Collins, and A. Schweiger. Funding was provided by NASA Polar Programs and by the NASA EOS interdisciplinary project POLES.

## QUESTIONS?

*FluxNet* may be obtained via anonymous ftp as described in the next section. If you have questions or bug reports, contact:

Jeff Key
NOAA/NESDIS
1225 West Dayton Street
Madison, WI 53706
e-mail: jkey@ssec.wisc.edu

Are you on the mailing list? If you have requested information via e-mail then you are. If not, and you plan to use *FluxNet*, please register on the web at **http://stratus.ssec.wisc.edu**.

Continued work on this program is largely unfunded. I'll be happy to answer questions about things that are not in the User's Guide, and will fix bugs in a reasonably short period of time. Keep in mind, however, that I may not be able to provide an immediate response.

## DISCLAIMER

This program is distributed as "freeware". Except when otherwise stated in writing the program is provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. In no event unless agreed to in writing will the author or any other party who may modify and/or redistribute the program be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program.

## A NOTE ON "CAUTIONS"

There are a number of cautions given in this manual. These are not meant to discourage you from using *FluxNet*, but rather to inform those users with little experience in radiative transfer or neural networks of things that this model does not do well. Ultimately you have to decide if what you are trying to do is reasonable, but these cautions should help steer you away from those aspects of the model that have the greatest uncertainty.

## REFERENCING *FLUXNET* IN PUBLICATIONS

While *FluxNet* is a unique tool, it alone does not expand our knowledge of radiative transfer, and has therefore not been published in a refereed journal. So, if you are presenting results obtained using *FluxNet*, how do you reference it? If you want a single reference, use

Key, J. and A.J. Schweiger, 1998, Tools for atmospheric radiative transfer: Streamer and FluxNet, *Computers & Geosciences*, 24(5), 443-451.

or simply reference this *User's Guide*:

Key, J., E. Amano, J. Collins, and A. Schweiger, 2001, FluxNet User's Guide, Cooperative Institute for Meteorological Satellite Studies, University of Wisconsin, 1225 West Dayton St., Madison, WI, 27 pp.

## WHAT'S NEW

Version 4 uses profile information specified as sigma levels rather than pressure levels. This allows for elevated surfaces in simulations. Cloud temperature has replaced cloud pressure in the input stream, making the program more useful in some satellite applications. The training data now includes many colder and higher elevation polar profiles. FluxNet was trained on data from Streamer v3.0, the most recent release.

# 2 OBTAINING AND BUILDING *FLUXNET*

**IMPORTANT!**

*If you download the code, please register on the web at* **http://stratus.ssec.wisc.edu**. *Doing so will ensure that you are kept informed, via email, of bug fixes and updates.*

*FluxNet* may be obtained for implementation under UNIX and other operating systems. With a few exceptions the source code is the same for all, but the method of getting the files and building the executable program varies. The C programs are very short and generic, and should run on any system with a C compiler.

## USER'S GUIDE

The *User's Guide* can be obtained via anonymous ftp as described below. The files are **userman.ps** (Postscript) or **userman.pdf** in the **docs** subdirectory of the main **fluxnet** subdirectory. You can view the files with a postscript viewer such as Ghostview/Ghostscript and with Adobe Acrobat Reader, respectively. Because of size limitations with some systems, you may need special printing options; e.g., **lpr -s userman.ps** on a UNIX machine. To get the manual via anonymous ftp:

1. **ftp stratus.ssec.wisc.edu**
2. Use **anonymous** as the username and your e-mail address as the password
3. **cd pub/fluxnet/docs**
4. **bin**
5. **get userman.ps** or **userman.pdf**
6. **quit**

## PROGRAMS AND SOURCE CODE

There are two distributions of *FluxNet*: **fnet.tar.gz** (or **fnet.zip** for MS Windows) contains only the main programs. The other, **fnetall.tar.gz** (or **fnetall.zip**), contains everything in **fnet.tar.Z** plus all the programs used to develop *FluxNet*. You'll only need the latter if you intend to implement your own neural network and you'll only need the former if you don't. The executables (binaries) for various operating systems are not included in these files.

On your computer, create a directory for *FluxNet* and switch to it. *FluxNet* can then be obtained via anonymous ftp as follows:

1. **ftp stratus.ssec.wisc.edu**
2. Use **anonymous** as the username and your e-mail address as the password
3. **cd pub/fluxnet**
4. **bin**
5. **get fnet.tar.gz** or **fnetall.tar.gz**
6. **quit**

The executables for UNIX machines (Sun Solaris, SGI Irix, and Linux on PCs) are available in the directory **pub/fluxnet/bin**. They must be downloaded separately.

Next, uncompress the tar file and extract the program files:

**gzip -d fnet.tar.gz**
**tar xvf fnet.tar**

You can then delete the file **fnet.tar**. Follow the instructions below to compile the programs, if necessary.

## BUILDING THE PROGRAM (IF NECESSARY)

If you need to build the executables, follow these steps. The following example is for compiling **fluxnet** (or **fluxnet.exe**). The procedure is the same for **fluxnetp** (using **netp**). Compile the main program **fluxnet.c** and the network file **net.c** (generated by SNNS):

| | |
|---|---|
| **cc -O fluxnet.c net.c -lm -o fluxnet** | **(basic Unix; use "gcc" for Linux)** |
| **bcc fluxnet.c net.c** | **(Borland C; use "cl" for Visual C)** |

You should now have the executable file **fluxnet** or **fluxnet.exe**.

## DIRECTORY STRUCTURE

The full distribution contains the following directories:

fluxnet

| | |
|---|---|
| docs | - User's guide |
| source | - Source code, C and IDL |
| bin | - Binary executable files |
| streamer | - Programs and data for creating the training data with *Streamer* |
| lib | - *Streamer* program files for the subroutine library |
| snns | - C and Unix programs for preparing to run SNNS, analyzing training data and results |
| TOVSdata | - TOVS-derived profile data |
| testio | - Test input/output files |

# 3 RUNNING *FLUXNET*

There are two programs that correspond to the two trained networks: the network without temperature and humidity profiles (**fluxnet**) and the network that incorporates profiles (**fluxnetp**). MS Windows executables will have the **.exe** extension.

To run *FluxNet* type:

**fluxnet <input-file-name> <output-file-name>**

where **fluxnet** may be **fluxnetp**, and **test.in** and **test.out** are the names of input and output files, respectively (which can be any names).

For IDL (v4.0 and higher), the procedures in the **fluxnet.pro** and **fluxnetp.pr**o files must be compiled then executed. For example, from the IDL prompt:

**> .run fluxnet**
**> fluxnet,'test.in','test.out'**

The directory **testio** has test input data.

Table 1 lists the input required for the network that does not use atmospheric temperature and humidity profiles. Table 2 gives the input for the network with profiles. Table 3 lists the network output, which is the same for both networks. In the input and output descriptions, all variables are floating point unless otherwise indicated. See the test input files in the next section for examples. Each line in the input file should consist of a value for each variable, in the order shown.

**CAUTION**

- The recommended range of values for each variable refers to the range of values that was used to train *FluxNet*. The network will behave unpredictably for input values that fall outside of this range.
- Make sure you are using the correct units for each variable. If you do not know the value of a particular variable, **do not leave it blank**; rather, **enter the default value**.

Additionally, not all values within the specified ranges are necessarily reasonable. In some cases a variable value may be reasonable but in other cases it may not. There are dependencies between variables that must be considered. For example, water clouds would probably not exist during winter in the middle or upper troposphere in the Arctic, so specifying a cloud top pressure of 400 mb when the surface temperature is 240 K, implying winter conditions, will result in large errors in the longwave fluxes. This happens because *FluxNet* was trained with reasonable values; e.g., water cloud temperatures no less than 253 K, ice cloud optical depths less than 50, etc.

**Table 1:** Input for the network **without profiles** (**fluxnet**).

| | |
|---|---|
| *tsurf* | Surface (skin) temperature, in degrees Kelvin |
| *emissurf* | Surface emissivity, recommended range 0.91 – 1.0 |
| *surftype* | Surface type indicator: 1=open sea water, 3=snow/ice, 5=generic vegetation (2 and 4 are not used). |
| *albsurf* | Surface shortwave broadband albedo. This is the albedo given the atmospheric (including cloud cover) conditions, not the inherent, no-atmosphere albedo as in *Streamer*. In other words, it is the albedo that would be measured by up- and down-looking radiometers. The recommended range for each surface type is 0.25 – 0.99 (snow), 0.1 – 0.25 (water), and 0.15 – 0.55 (vegetation). For nighttime conditions, set *albsurf* to 0. |
| *cldphase* | Cloud particle phase. Enter 0 for liquid water cloud, 1 for ice cloud (solid hexagonal columns). |
| *cldre* | Cloud particle effective radius in microns. Recommended ranges: 2.5 – 20 μm for water (liquid) clouds, and 10 – 80 μm for ice clouds. |
| *cldwc* | Cloud water content in g $m^{-3}$. Recommended ranges: 0.05 - 0.5 g $m^{-3}$ for water (liquid) clouds, and 0.0007 - 0.11 g $m^{-3}$ for ice clouds. |
| *cldtau* | Cloud visible optical depth. Recommended range for water cloud is 0 – 150 (0 is clear); the ice cloud range is 0 - 50. |
| *cldtemp* | Cloud top temperature (K). Use values less than 273 K for ice clouds and 253 K or greater for water (liquid) clouds. |
| *cldfrac* | Cloud fraction, range 0.0 – 1.0. For clear sky set *cldfrac* to 0, *cldphase* to 0, *cldre* to 10, *cldwc* to 0.2 *cldtemp* to 273, and *cldtau* to 0. |
| *zen* | Solar zenith angle in degrees. Recommended range 0 - 90. Use 90 for nighttime (dark) conditions. |
| *tauhaze* | Aerosol optical depth (unitless), recommended range 0.05 – 0.5 |
| *o3* | Total column ozone in g $m^{-2}$, recommended range 6.4 – 9.7l.<br>(1 Dobson unit = 0.021416667 g $m^{-2}$) |
| *h2ocol* | Total column water amount (water path) in g $m^{-2}$, recommended range 700 – 70,000. |

**Table 2:** Input for the network **with profiles** (**fluxnetp**).

| | |
|---|---|
| *tsurf* | Surface (skin) temperature, in degrees Kelvin |
| *emissurf* | Surface emissivity, recommended range 0.91 – 1.0 |
| *surftype* | Surface type indicator: 1=open sea water, 3=snow/ice, 5=generic vegetation (2 and 4 are not used). |
| *albsurf* | Surface shortwave broadband albedo. This is the albedo given the atmospheric (including cloud cover) conditions, not the inherent, no-atmosphere albedo as in *Streamer*. In other words, it is the albedo that would be measured by up- and down-looking radiometers. The recommended range for each surface type is 0.35 – 0.9 (snow), 0.1 – 0.2 (water), and 0.15 – 0.55 (vegetation). For nighttime conditions, set *albsurf* to 0. |
| *cldphase* | Cloud particle phase. Enter 0 for liquid water cloud, 1 for ice cloud (solid hexagonal columns). |
| *cldre* | Cloud particle effective radius in microns. Recommended ranges: 2.5 – 20 μm for water (liquid) clouds, and 10 – 80 μm for ice clouds. |
| *cldwc* | Cloud water content in g $m^{-3}$. Recommended ranges: 0.05 – 0.5 g $m^{-3}$ for water (liquid) clouds, and 0.0007 - 0.11 g $m^{-3}$ for ice clouds. |
| *cldtau* | Cloud visible optical depth. Recommended range for water cloud is 0 - 150 (0 is clear); the ice cloud range is 0 - 50. |
| *cldtemp* | Cloud top temperature (K). Use values less than 273 K for ice clouds and 253 K or greater for water (liquid) clouds. |
| *cldfrac* | Cloud fraction, range 0.0 – 1.0. For clear sky set *cldfrac* to 0, *cldphase* to 0, *cldre* to 10, *cldwc* to 0.2 *cldtemp* to 273, and *cldtau* to 0. |
| *zen* | Solar zenith angle in degrees. Recommended range 0 - 90. Use 90 for nighttime (dark) conditions. |
| *tauhaze* | Aerosol optical depth (unitless), recommended range 0.05 – 0.5 |
| *o3* | Total column ozone in g $m^{-2}$, recommended range 6.4 – 9.7.<br>(1 Dobson unit = 0.021416667 g $m^{-2}$) |
| *t_in(i), i* = 1..*nlevs*<br>*wv_in(i), i* = 1..*nlevs* | Temperature in degrees Kelvin and water vapor mixing ratio in g/kg. The levels (*nlevs*=20) correspond to the following sigma coordinates:<br><br>0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50,<br>0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 1.00<br><br>where the sigma coordinate is the pressure at some level divided by the surface pressure. The sigma coordinate for the surface is always 1. For example, if the surface pressure is 1013 mb the sigma coordinate for the 500 mb level it is 0.494; for the 350 mb level it is 0.346, etc. |

**Table 3:** Output for both **fluxnet** and **fluxnetp**.

| | |
|---|---|
| *swdsrf* | Total (direct plus diffuse) downwelling surface shortwave flux, in W $m^{-2}$ |
| *swusrf* | Upwelling surface shortwave flux, in W $m^{-2}$ |
| *lwdsrf* | Downwelling surface longwave flux, in W $m^{-2}$ |
| *lwusrf* | Upwelling surface longwave flux, in W $m^{-2}$ |
| *swdtoa* | Downwelling top-of-atmosphere shortwave flux, in W $m^{-2}$ |
| *swutoa* | Upwelling top-of-atmosphere shortwave flux, in W $m^{-2}$. See note below. |
| *lwutoa* | Upwelling top-of-atmosphere longwave flux, in W $m^{-2}$ |

## NOTE ON TOA FLUXES

While the TOA downwelling shortwave flux is calculated by *FluxNet*, it will generally be more accurate to calculate it as the solar constant times the cosine of the solar zenith angle. The solar constant used in *Streamer* for generating the training data was 1354.2 W $m^{-2}$, which is based on the mean Earth-Sun distance for the spectral band 0.28 - 4.0 µm. For example, with a 50 degree solar zenith angle, the TOA downwelling shortwave flux is 870.46 W $m^{-2}$. The downwelling longwave flux at TOA is essentially zero.

The cloud radiative effect, more commonly called "cloud forcing", can be computed from the net shortwave and longwave fluxes at the surface and TOA. It is defined as

$$CF_{\lambda,z} = \int_0^{A_c} \frac{\partial F_{\lambda,z}}{\partial a} da = F_{\lambda,z}(A_c) - F_{\lambda,z}(0)$$

where $F_{\lambda,z}$ is the net flux (W $m^{-2}$) for shortwave or longwave radiation ($\Lambda$) at either the surface or TOA ($z$), and $A_c$ is the cloud fraction in the scene. The net flux is equal to the downwelling minus the upwelling fluxes. Analogous to net radiation, the all-wave net cloud forcing at either the surface or TOA can be calculated from

$$CF_z = CF_{shortwave} + CF_{longwave}$$

# 4 SAMPLE INPUT AND OUTPUT

Examples of input data are given below. Note that you can format the input with any number of spaces, tabs, or carriage returns between data values.

## INPUT: WITHOUT PROFILES

```
270.05 0.954 5. 0.397 1.  56.3 0.107   4.1  219.9 0.97 27.67 0.17  6.647  6306.54
271.17 0.966 1. 0.106 0.  11.9 0.103   7.0  260.4 1.00 60.85 0.42 10.497  6103.54
276.77 0.968 1. 0.087 0.  15.4 0.079   0.9  257.1 0.42 20.61 0.24  8.172  9481.55
```

## INPUT: WITH TEMPERATURE & RELATIVE HUMIDITY PROFILES

```
288.85 0.965 5. 0.000 0.  10.0 0.200   0.0  273.0 0.00 90.00 0.06  6.771
217.57 218.13 221.05 226.71 233.14 237.17 235.12 240.35 246.78 252.50
257.57 262.51 267.33 271.32 274.55 277.58 281.26 285.28 288.50 288.85
 0.0020  0.0057  0.0192  0.0446  0.0871  0.1501  0.2401  0.3588  0.5135  0.7269
 0.8437  0.9145  1.0487  1.4036  2.1436  3.5965  6.5303  7.4423  7.9373  8.7263
280.05 0.944 1. 0.052 0.  10.0 0.200   0.0  273.0 0.00 52.13 0.12  9.132
218.94 212.79 207.39 212.49 224.35 235.09 243.30 249.41 254.45 259.01
263.28 267.15 270.52 273.06 275.48 277.76 278.24 275.26 276.55 280.05
 0.0020  0.0095  0.0323  0.0748  0.1464  0.2523  0.4084  0.6011  0.7744  0.9580
 1.1259  1.3415  1.6184  1.8148  2.0094  2.1955  2.5268  3.4655  4.5085  4.8212
258.35 0.912 8. 0.136 0.  11.2 0.404   0.2  258.3 0.46 49.18 0.48  7.389
202.95 205.60 206.28 203.19 213.16 223.13 231.73 238.82 245.11 250.90
255.32 258.11 259.88 262.15 261.69 260.01 259.04 259.95 259.75 258.35
 0.0020  0.0032  0.0107  0.0247  0.0483  0.0833  0.1970  0.3724  0.6320  0.9912
 1.3174  1.3624  1.1444  1.1054  1.0411  0.8809  0.5263  0.5880  0.6422  0.5647
```

## OUTPUT

The following sample output is the same format for both **fluxnet** and **fluxnetp**, with these variables on each line: surface downwelling shortwave, upwelling shortwave, downwelling longwave, and upwelling longwave; TOA downwelling shortwave, upwelling shortwave, and upwelling longwave fluxes. The sample data below does not correspond to the input data shown above.

```
 897.29   154.57   400.57   441.09  1238.49   237.55   241.95
  -2.30    -0.92   380.20   442.45     2.92    -2.84   250.49
   2.05    -0.51   264.08   327.27     2.59    -4.45   236.25
```

# 5 SAMPLE APPLICATIONS

*FluxNet* can be used in any application that requires fast and accurate radiative transfer calculations. Simple parameterizations are fast but most cloud, atmospheric, and surface properties are implicit and cannot be controlled by the user. Radiative transfer models are flexible and accurate, but are not usually fast enough for processing very large data sets.

*FluxNet* is currently being used in CASPR (http://stratus.ssec.wisc.edu/caspr) to compute radiative fluxes based on cloud and surface properties estimated from AVHRR data and to compute radiative fluxes with the International Satellite Cloud Climatology Project (ISCCP) D1 cloud data set. An example of the latter is shown in Figure 1.

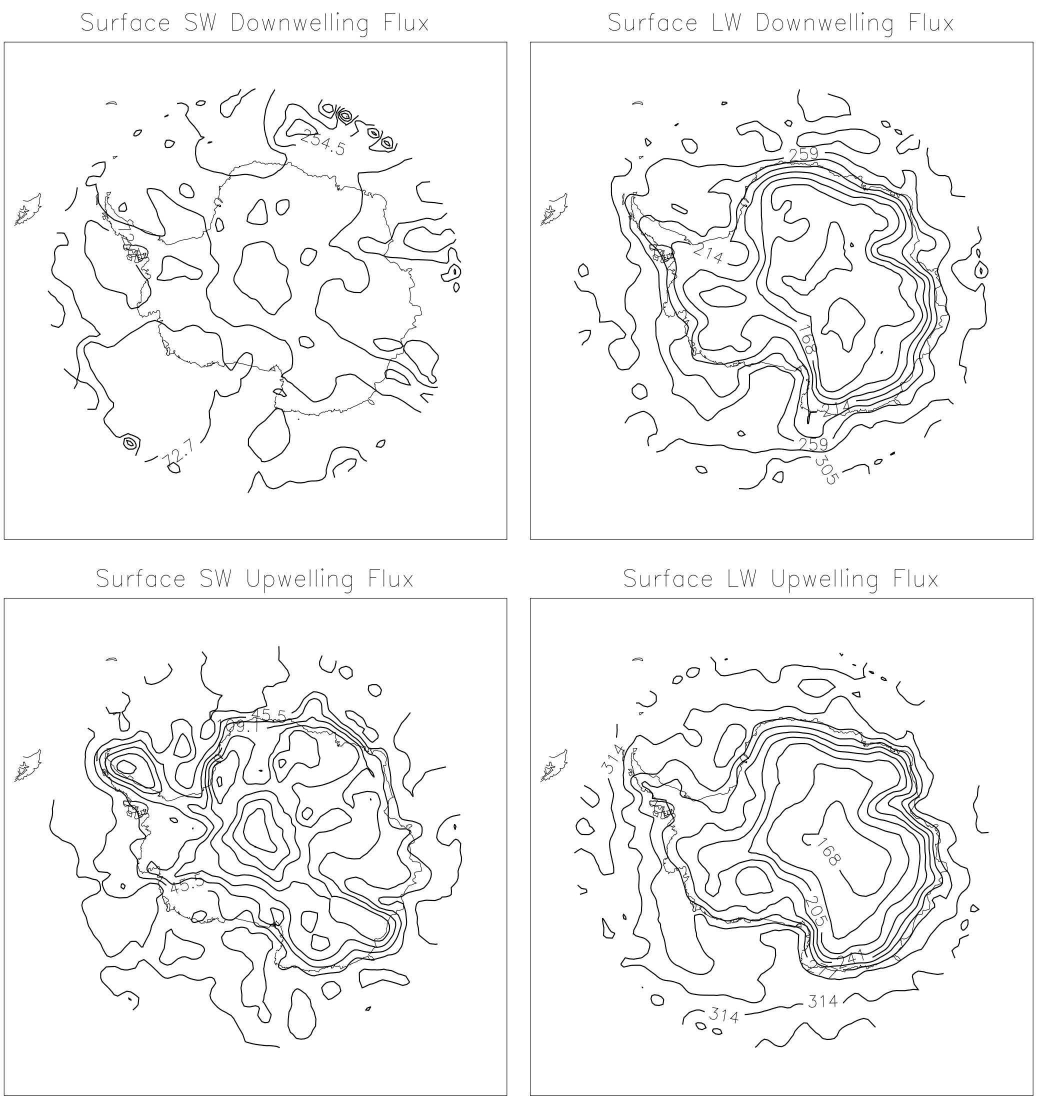



**Figure 1.** Downwelling and upwelling shortwave and longwave fluxes at the surface (W m$^{-2}$) over the Arctic on June 1, 1986, as computed using the ISCCP D1 cloud product and *FluxNet*.

Figure 2 gives an example of using FluxNet to calculate downwelling shortwave and longwave fluxes at the surface over a range of cloud amounts and cloud optical depths.

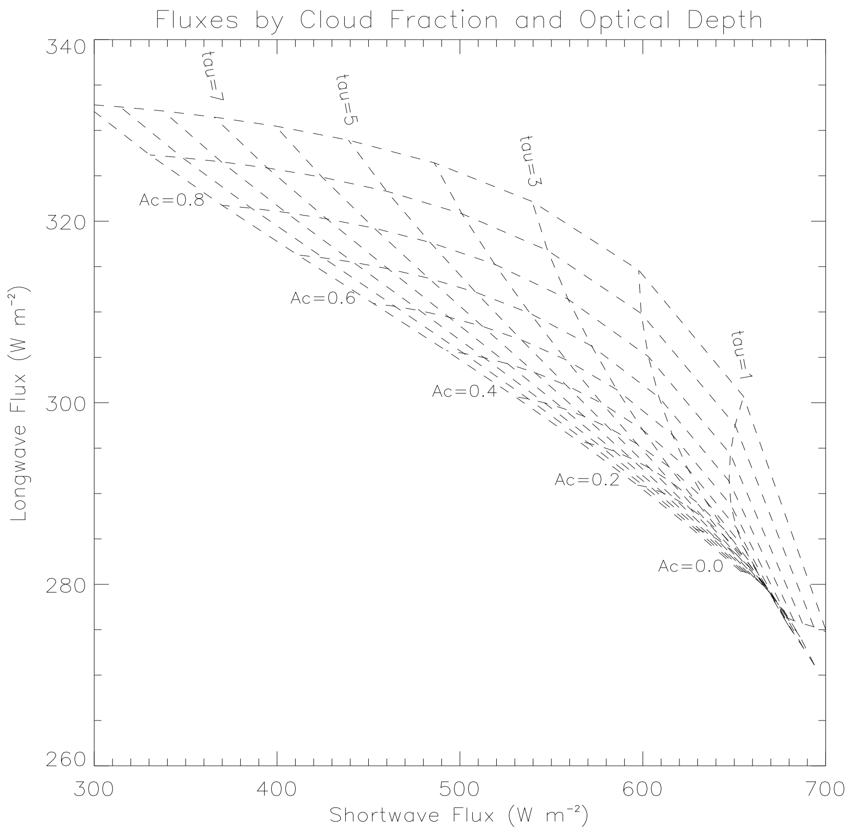



**Figure 2.** Downwelling shortwave and longwave fluxes at the surface as a function of cloud amount (Ac) and cloud visible optical depth (tau). The results are for a solar zenith angle of 50 degrees, a surface albedo of 0.15, and a water cloud with an effective radius of 10 microns.

# 6 TECHNICAL NOTES

This section gives a brief theoretical background for those unfamiliar with neural networks, and some information that may be of assistance to those considering modifying *FluxNet*. What it does *not* attempt to do is to teach those unfamiliar with ANNs how to create an artificial neural network. Interested users should refer to introductory texts on ANNs before attempting to modify *FluxNet*.

*FluxNet* was trained using the *Stuttgart Neural Network Simulator, version 4.1* (University of Stuttgart, Institute fior Parallel and Distributed High Performance Systems, Germany). Information about SNNS can be found on the World Wide Web at **http://www.informatik.uni-stuttgart.de/ipvr/bv/projeckte/snns/snns.html.**

## WHAT IS A BACKPROPAGATION NETWORK?

Artificial neural network refers to a family of computational and pattern-recognition algorithms typically consisting of a group of interconnected processing nodes. ANNs were initially used by neuroscientists in an attempt to understand certain functions of the brain, hence the misleading moniker. Over the past decade they have increasingly been applied to tasks involving the recognition of complex patterns such as signal processing, optical character recognition, and even stock market forecasting. Although a variety of ANN architectures has been created, the three- and four-layer backpropagation networks are the most popular.

The backpropagation network is what is called a multi-layer feed-forward network. The signals from the input units are fed forward through processing nodes in the hidden layers to the output units; the output is then compared to desired results, and the error is propagated backwards from the output layer through the hidden layers (hence the name “backpropagation”) and the weight of each connection is adjusted accordingly.

The input to each processing unit *j* is a weighted sum of the output from the units in the previous layer. The output from this unit *j* is a function, called the activation function, of this sum:

$$o_j = f\left(\sum_i w_{ji} \cdot o_i\right), \tag{1}$$

where $w_{ji}$ is the weight of the connection from input unit *i* to output unit *j* and $o_i$ is the signal from input unit *i*.

There are several popular activation functions, such as the sigmoid (or logistic function)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

and the hyperbolic tangent function. As well as adding a non-linear element to the network, these “squashing” functions output values in the range [0, 1] (for the sigmoid function) or [-1, 1] (for the *tanh* function), thus normalizing the outputs and preventing them from blowing up into very large numbers.

Once the input signals are propagated forward through the hidden layers to the output layer, the output is compared to the desired results for the input pattern. The error, $E_k$, at each output unit $k$ is the difference between the desired output ($t_k$) and the actual output ($o_k$):

$$E_k = t_k - o_k . \tag{3}$$

A fundamental problem for a multi-layer network is how to update the weights of the network connections. Rumelhart *et al.* (1986) formulated an algorithm called the generalized delta rule as a solution to this problem. Weight updates in the generalized delta rule is a function of the error signal $\delta_i$ at unit $i$ and the output $o_j$ from the preceding unit $j$, the unit sending the signal:

$$\Delta w_{ji} = \eta \delta_j o_i , \tag{4}$$

where $\eta$ is the "learning rate," and $\delta$ is calculated slightly differently for output and hidden units. For output units,

$$\delta_k = E_k \cdot f'(net_k) \tag{5}$$

and for hidden units,

$$\delta_j = f'(net_j) \cdot \sum_k \delta_k w_{kj} , \tag{6}$$

where $f'(net_j)$ is $o_j(1 - o_j)$ for the sigmoid function.

In other words, the learning cycle in a backpropagation network involves the repetitive simultaneous presentation of matching input and output patterns while the weights are adjusted using a gradient descent search. Thus a neural network can also be viewed as a non-linear numerical optimization procedure.

Characteristics of neural networks that make them attractive for the research presented here are: (1) the four-layer network can, theoretically, determine any computable function, (2) no assumptions about the statistical distribution of input variables are made, and (3) they are very fast once they are trained. However, since neural network-based estimation methods do not include any assumptions about the underlying non-linear physics, estimates can only be truly optimal with respect to the training data set and estimation errors need to be determined through the application to an independent test or validation data set.

## *FLUXNET* ARCHITECTURE

The "optimal" network architecture is dependent on any given data set, but can be determined relatively painlessly through some well-established heuristics. Which architecture works and which doesn't can happen quite capriciously. Generally, you can start with a very small network and keep adding more nodes until the network starts performing acceptably well. Alternatively, you can start with a big network (e.g., two hidden layers with thirty nodes each) and keep pruning away the nodes until the network starts behaving badly. In the end, you want the smallest network that will get the job done, since bigger networks require more calculations and tend to "memorize" the training data rather than generalizing the function.

The *FluxNet* network architectures are given in Table 1.These network architectures were found to produce reasonably good approximations of the expected output data. This version of *FluxNet* was trained with *Streamer* v3.0.

**Table 4:** Network architectures for the versions without profiles (**fluxnet**) and with profiles (**fluxnetp**).

| | Input Nodes | Hidden Layer 1 Nodes | Hidden Layer 2 Nodes | Output Nodes | # of Training Cycles | Mean Square Training Error |
|---|---|---|---|---|---|---|
| **fluxnet** | 14 | 16 | 10 | 7 | 7,000 | 0.0008 |
| **fluxnetp** | 53 | 24 | 12 | 7 | 8,000 | 0.0004 |

## DATA USED TO TRAIN *FLUXNET*

Figures 2 and 3 show the relative frequencies and ranges of the input and target (or output flux) variables used to generate data (*Streamer*) to train *FluxNet*. Figure 4 shows the temperature and humidity profiles used to generate the target fluxes with *Streamer.* Figure 5 give the relative frequencies of the latitudes of these profiles. While the profiles do cover all latitudes, the midlatitudes are better represented than the tropics. The profiles were interpolated to the 20 pressure levels given in Table 2 before training **fluxnetp**.

## *FLUXNET* TRAINING ERRORS

*Streamer* was run with approximately 17,000 sets of randomly generated input data. The nonprofile *FluxNet* network, **fluxnet**, was trained with total column precipitable water rather than the water vapor profile. Comparisons of **fluxnet** outputs with the corresponding *Streamer* output are shown below, in Figure 6. Deviations of *FluxNet* from *Streamer* results are also given in the figure as bias (*FluxNet* minus *Streamer* fluxes) and root-mean-squared errors. Errors for **fluxnetp** are a few W $m^{-2}$ smaller than for **fluxnet**.

## COMPUTATION SPEED

*FluxNet* is 2 to 4 orders of magnitude faster than *Streamer*. It took *Streamer* took 24 hours on a 800 MHz PC (Linux) to calculate the longwave training data. It took *FluxNet* 15 seconds to calculate the same data. Training time ranges from 2-24 hours, depending on the size of the training data set and the number of nodes in the network.

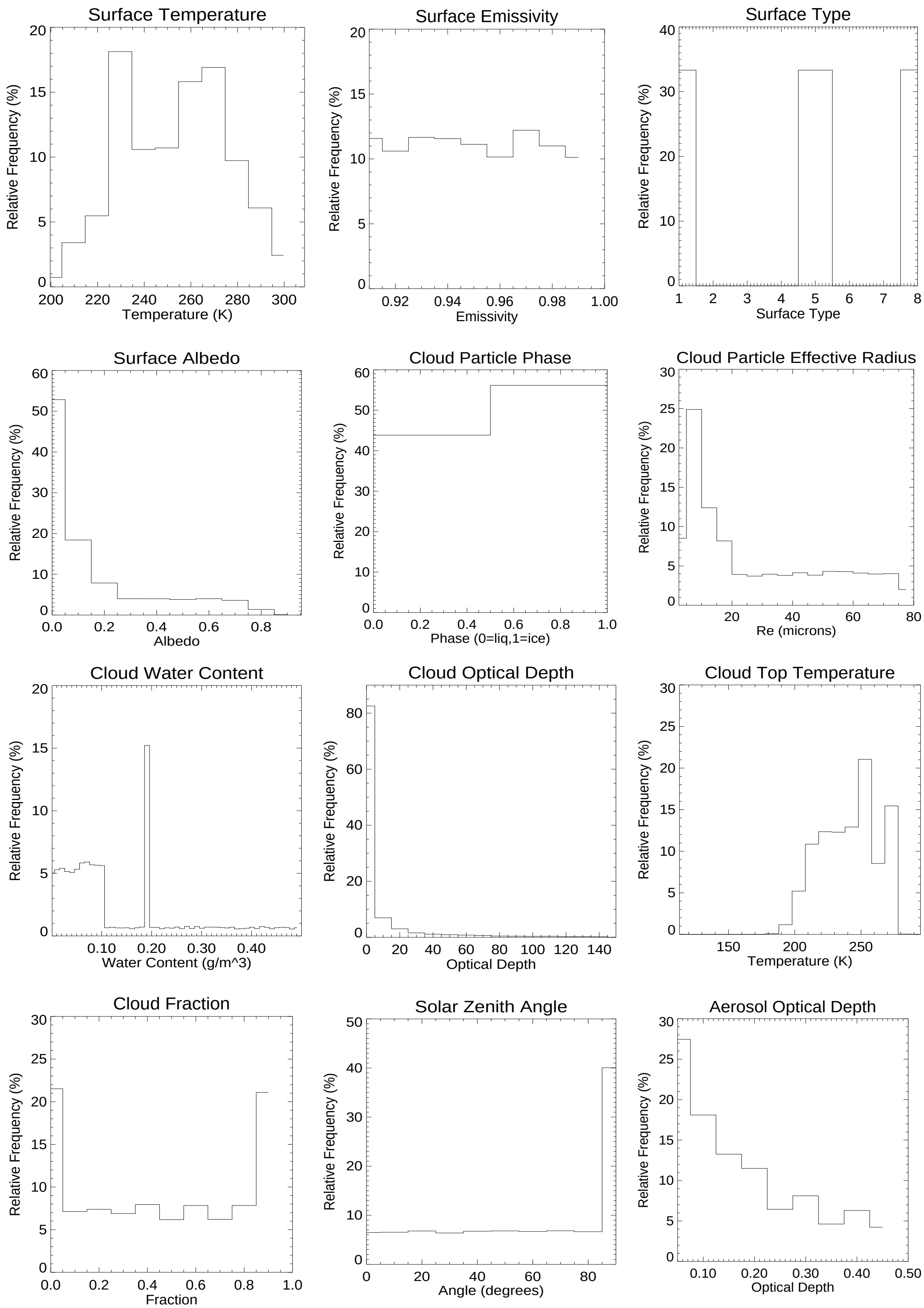



**Figure 3.** Relative frequencies of input variables used to train *FluxNet*. Spikes in the histograms of cloud parameters correspond to their values for clear cases.

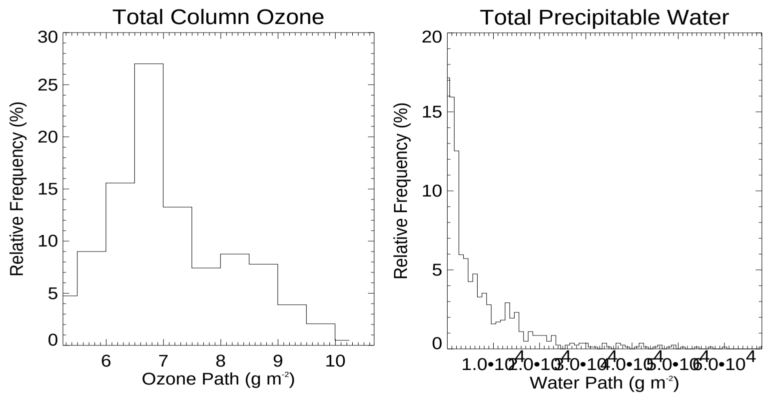



Figure 2, continued.

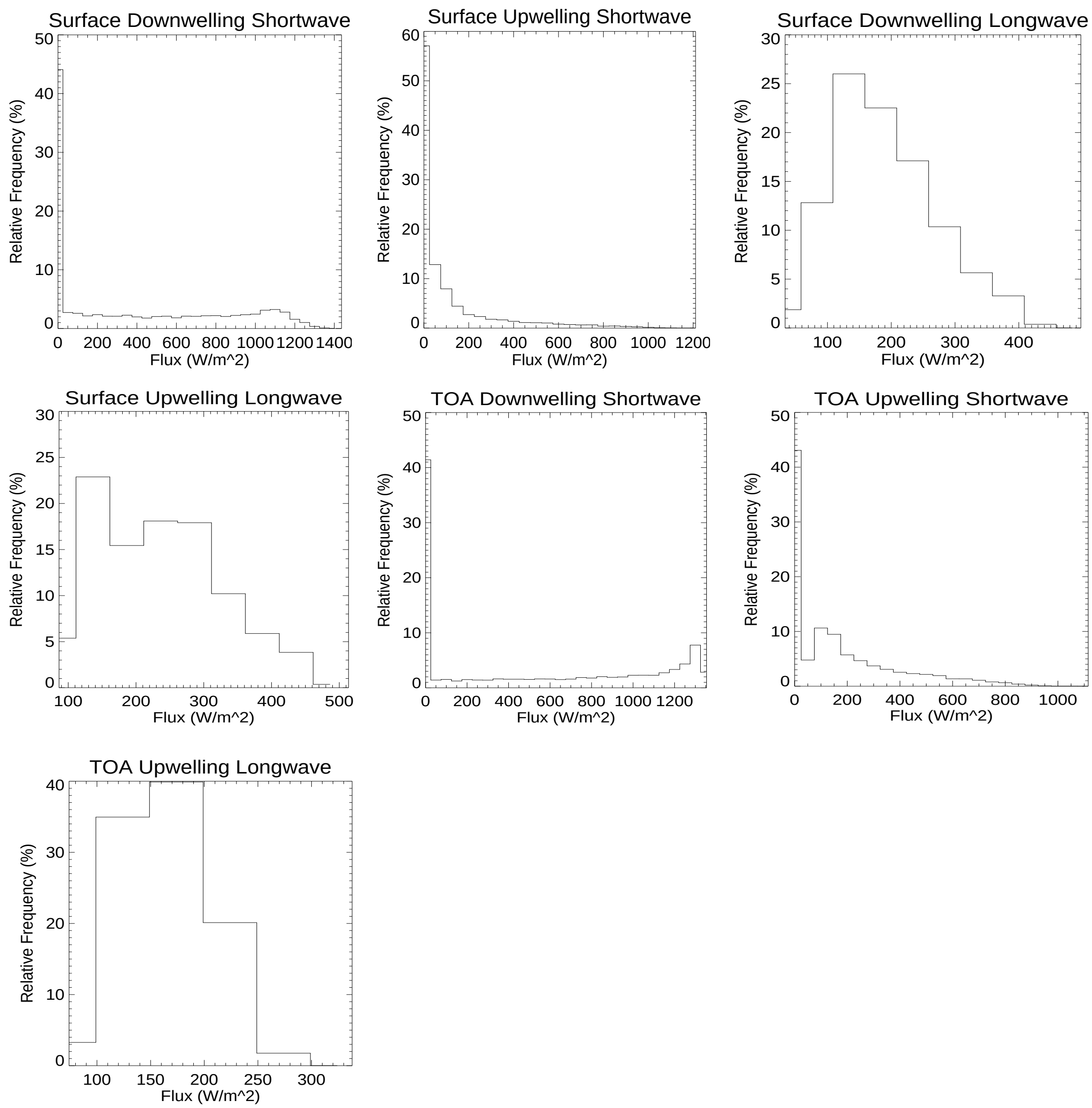



**Figure 4.** Relative frequencies of target (output) variables used to train *FluxNet*.

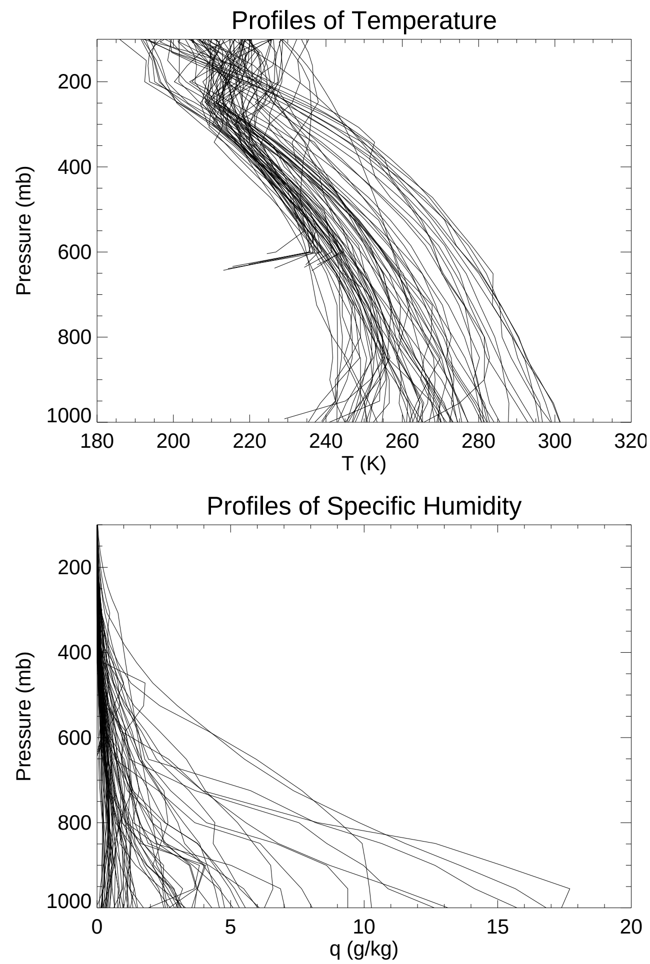



**Figure 5.** Sample temperature and humidity profiles used in generating training and testing data for *FluxNet*.

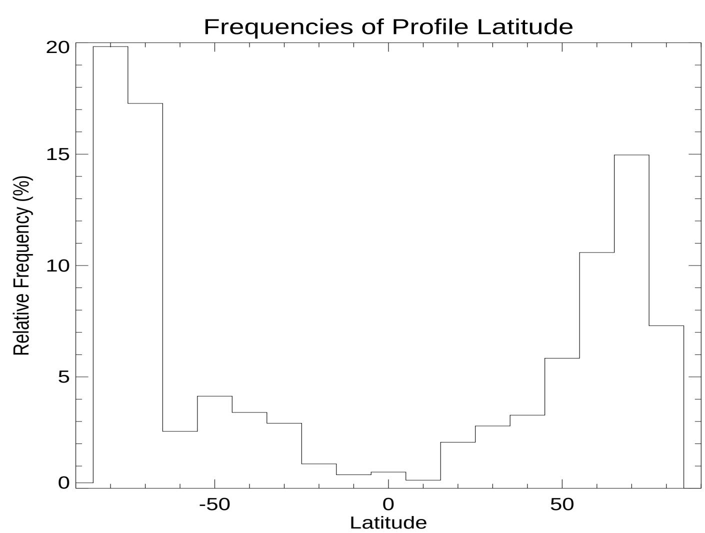



**Figure 6.** Relative frequencies of the latitude of profiles used in training.

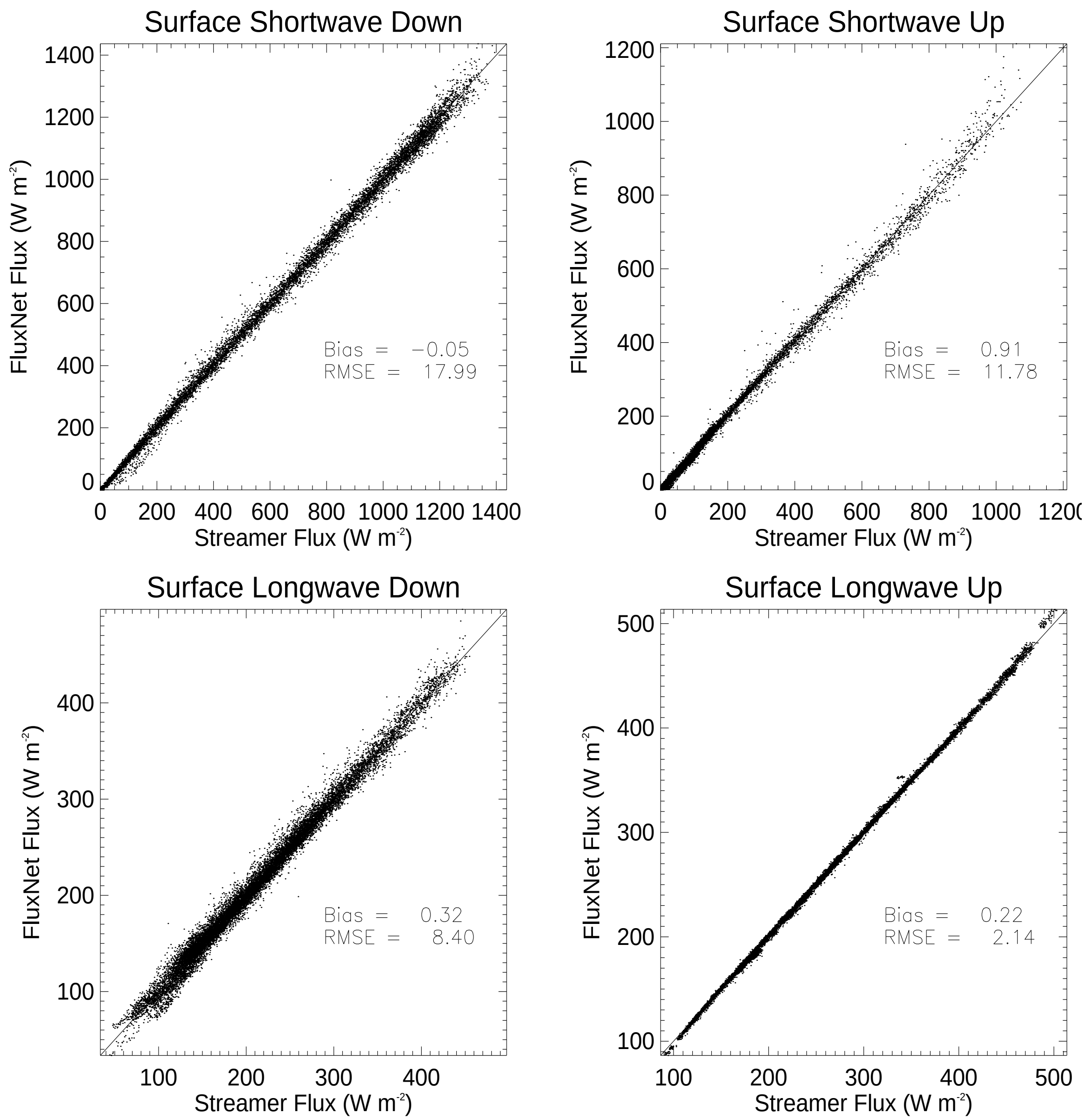



**Figure 7.** Comparison of *FluxNet* (no profiles) and *Streamer* fluxes for one test data set. Other test data sets yield almost identical results.

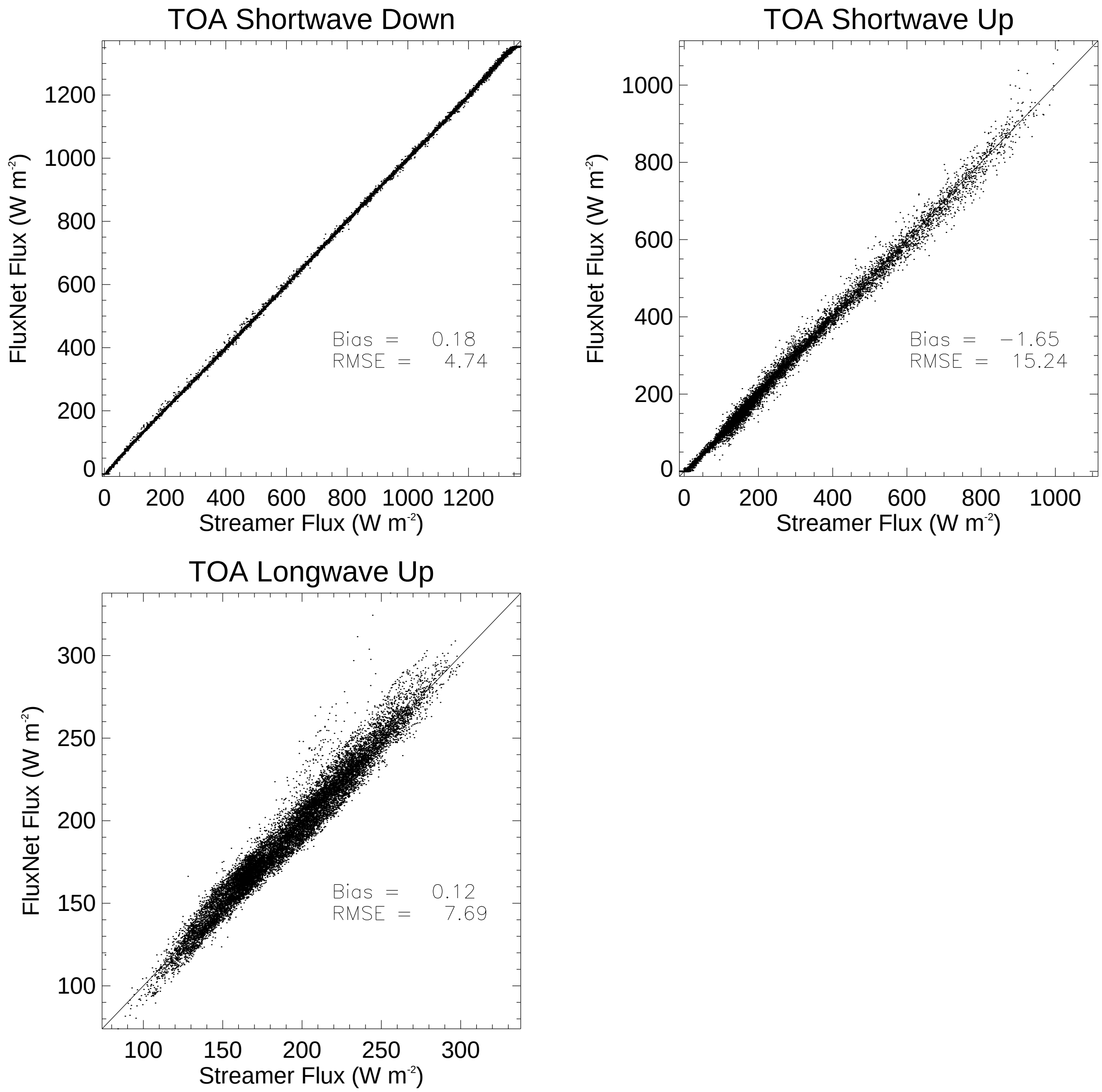



Figure 6, cont.

# 7 CUSTOMIZING *FLUXNET*

If the network configurations that are currently available do not suit your needs, then you will have to develop your own. Users who wish to modify or customize *FluxNet* should have some practical knowledge of neural networks, Fortran, C, and Unix programming. The major steps involved in designing and implementing a new network are:

1. **Generate training and testing data** by either sampling your own dataset or running *Streamer.*
2. Reformat and **scale** the training data.
3. **Train the network** with the SNNS software.
4. **Create the main C and/or IDL program** to use the C code generated by SNNS and build the executables.

## GENERATING TRAINING DATA

The training dataset is the most important part of customizing your network. It involves the selection of appropriate input and output variables and the selection of cases that are representative of the entire dataset to which the final network will be applied. In selecting the input variables, avoid redundancy. For example, if cloud particle effective radius will have only two values, one for liquid cloud and one for ice cloud, then having a cloud phase indicator variable would be redundant.

In selecting the training cases it is extremely important that the training set be representative of not only the range of conditions in the full dataset, but also the approximate frequency distributions of the individual variables. This is perhaps easiest to accomplish by sampling a very large dataset and then using *Streamer* to compute the fluxes. Another possibility is to use the version of *Streamer* supplied with the *FluxNet* distribution, which has a special routine to randomly generate values of your input variables. This procedure is described next.

For generating training and testing data for *FluxNet*, *Streamer* was run as a subroutine. All programs are written in Fortran. While the programs and data used to build *FluxNet* are provided as part of the distribution, the subroutine library will need to be built. (If you are interested in obtaining the entire *Streamer* model, go to the World Wide Web page at **http://stratus.ssec.wisc.edu**.)

The subroutine implementation of *Streamer* allows greater flexibility in generating training data than the standard input file format. In the main program a range of values for each *FluxNet* input variable (e.g., surface albedo, cloud optical depth) is specified, and a uniform random number generator is employed to select a value within the range. Fluxes are then calculated for any number of cases over any number of profiles.

The files that will need to be modified are in the **fluxnet/streamer** subdirectory. You may want to change the *Streamer* main calling program and/or the output subroutine. These are **run.f**, and **write.f**, respectively. Note that the temperature and humidity profiles are written in **run.f** but the other variables are written by **write.f**. You may also need to change the default values set in **lib/defaults.f**. A Sun Solaris makefile is provided for compiling the program the procedures in **lib/** into a library. It may need to be modified for other operating systems. This library must be linked to the object code corresponding to **run.f** and **write.f**.

## SCALE THE TRAINING DATA

Before training the neural network the training and testing data must be scaled and SNNS pattern files must be created. The proper scaling must be determined such that each input and output variable value presented to neural network falls between 0 and 1 (actually, something less than one is better). The simplest scaling method is to subtract the minimum value from each value and divide by the observed range. The IDL program **hists.pro** (and **histsp.pro**) in the **snns** subdirectory plots a histogram and prints out the minimum, maximum, and range values for each variable.

Next sample and scale the raw input data, creating training and testing files with the program **mkdatfile**. The source code is **mkdatfile.c** (and **mkdatfilep.c**), which will need to be edited (scaling, number of variables, etc.) and compiled. This program is used as follows:

**mkdatfile invarfln outvarfln outfile modulo remainder**

where *modulo* and *remainder* are used for subsampling cases. For example, case i is selected if the remainder of i divided by *modulo* is *remainder*. *invarfln* and *outvarfln* are the files containing the raw input and output variables for network training. *outfile* is the file that will contain the scaled input and output. Of course, if you change the nature of the training data you must modify the C source code for this program. For example,

**mkdatfile runin.out runout.out run.trn 10 1**
**mkdatfile runin.out runout.out run.tst 5 2**

creates a scaled training file containing the 1st, 11th, 21st, etc. cases from the input files and a scaled test file with the 2nd, 7th, etc. input cases.

Now create SNNS pattern files for the training and testing data sets:

**mkpatfile run.trn > runtrn.pat**
**mkpatfile run.tst > runtst.pat**

**mkpatfile** and **mkpatfilep** (for patterns with profiles) are Unix scripts that add the required header to the training and testing data. The output files are the ones that will be used by SNNS. These scripts will need to be edited to indicate the proper number of cases and variables in your training and testing data files.

## TRAINING THE NETWORK WITH SNNS

The network can now be trained using either the graphical user interface (GUI) or the SNNS batch file facility.

> **CAUTION**
> Before training the network you should be familiar with running the SNNS neural network software. The instructions given here should be detailed enough to get you through, but that's about all! They apply to SNNS version 4.1.

**GUI**

1. Start SNNS.
2. Specify the network architecture.
   a) Click on BIGNET in the SNNS manager window then feed_forward.

   b) Click POS to set the display position to BELOW.

   c) Create a network with the appropriate number of input units, hidden units in the first hidden layer, hidden units in the second hidden layer, and output units. Make sure the type shown in the right-hand column of boxes is *Input*, then specify the number of input variables in the box corresponding to the *No. of units in the x-direction*. Specify 1 for the *No. of units in the y-direction*. Click ENTER. If the layer specification is acceptable then it will also appear in the left column of boxes.

   Click TYPE until Hidden is shown in the top box of the right column, then specify the number of hidden units for this first hidden layer in the *x-direction* box, and 1 for the *y-direction* box. Click ENTER. Specify the number of nodes for the second hidden layer, then click ENTER.

   Click TYPE until *Output* is shown in the top-right box, then specify the number of output nodes in the *x-direction* box, and 1 for the *y-direction* box. Click ENTER.

   d) Click FULL CONNECTION at the bottom of the window then CREATE NET then DONE.

5. Look at the network structure that you just defined by clicking the DISPLAY button in the main window (snns-manager).
6. LOAD the training and testing patterns via the FILE menu, clicking on PAT (patterns) to load the data. Click DONE.
7. Open the CONTROL window.
   a) set STEPS to 1

   b) set CYCLES to 100. You'll eventually need to go through 1,000 to 20,000 training cycles, but starting with 100 will give you a good idea of how well the network will train. You can click ALL again to continue training after the each set of 100 cycles.

   c) select **runtrn** with the first USE button. This button sets the training data file.

   d) select **runtst** with the second USE button. This button sets the test data file.

   e) with the first SEL.FUNC. button, choose *Std_backpropagation*

   f) with the second SEL.FUNC. button, choose *Topological_order*

   g) with the third SEL.FUNC. button, choose *Randomize_weights*

   h) set LEARN to 0.2 and 0.0

   i) click on SHUFFLE so that it is activated. This randomizes the order in which the training data set is presented to the network.

   j) click INIT to initialize (in this case, randomize) the weights of the connections.

   k) <u>don't</u> click DONE

12. To see the results of the test cases, enter a number of cycles next to VALID and then click that button. Click on GRAPH in the snns-manager panel to open up the graph window. This gives you a visual representation of how well the network is learning. SSE (sum of squared errors) will be plotted; you will probably need to change the y-axis range, either after the specified number of

cycles has been completed or after clicking STOP (click ALL to resume). Test results are shown in red on the plot.

13. In the SNNS control panel, click on ALL to start the training cycles. The command-line window where you started snns should start printing out a progress report. Generally, you want to train the network until the MSE goes below 0.001 (you may or may not be able to achieve this).
14. Once the network has been trained adequately, click FILE and save it with a name <network_name>.net.

*TIPS:* You'll get a better final result if you decrease the learning rate to 0.1 as the network gets closer to a solution. Watch out for overtraining and overfitting: the MSE for the training and test data should be about the same. Sometimes, the errors for the test data will start going up (not the small random fluctuations) even while the training error is going down. That means that the network is starting to overfit the data, although this is highly unlikely if the number of cases in the training file is large. Training typically takes from 6 to 24 hours (Sun Ultra 1, Model 170, 17000 cases each for training and testing). The standard backprop method with momentum may work better than standard backprop. Set the learning rate to 0.2 (the first of four value boxes) and the momentum to 0.5 (second box; leave the third and fourth boxes at the default of 0).

### SNNS Batch File

Edit the SNNS batch file **train.bat** (see the SNNS manual for details) and run it with the command

**batchman -q -f traintest.bat > err**

***Note: The batchfiles provided with FluxNet have not been thoroughly tested.***

## Examining the Results

From the *FILE -> RES* menu in SNNS the network generated outputs from the training or testing data can be saved to a file. Do not include the input patterns but do include the output patterns (the opposite of the default choices). To save results for only the test data you'll need to *USE* the appropriate data set in the *CONTROL* dialog window before saving the results file. Whatever file is being *USE*d will be fed through the network when the results file is saved. The IDL program **fluxnet/snns/plotres.pro** can be used, possibly without modification, to plot the difference between the target and output fluxes in these SNNS results files.

The test input data in **fluxnet/testio** were generated with the IDL program **fluxnet/snns/maketest.pro**. These data were then used to test the final networks. The IDL program **fluxnet/testio/plotdiff.pro** plots the difference between the *Streamer* fluxes and the fluxes generated by the final *FluxNet* programs.

## Conversion to C and IDL

The network file ***.net** can now be converted to C code using the SNNS command **snns2c:**

**snns2c snns-network-file net.c net**

The first argument to **snns2c** is the name of the network file that you saved in the SNNS session, e.g.,

**16_10.net**. The second argument is the name of the output network C file (with associated **net.h** file automatically generated). Using **net.c** is simple and clear. The third argument is the name given to the C function, i.e., the function that you will call to compute fluxes.

It can also be converted to an IDL procedure with the Unix script **snns2idl** in the **fluxnet/source** directory. You should first change the network name in the ***.net** file by editing the line that says “network name”, changing whatever is there (e.g., “16_10”) to simply “net”. Then convert it to IDL:

**snns2idl snns-network-file > idl-pro-file**

Neither the C nor IDL procedures converted from the SNNS network files handle data scaling or input/output so programs that do the scaling and call these procedures must be written. The **fluxnet.c** and **fluxnet.pro** files are the main programs for *FluxNet* and can be used as examples for your own design. The scaling factors must be taken from the **mkdatfile*.c** code, inverting the scaling equations for the output variables. The next section describes how to compile the C modules into an executable. For IDL (version 4.0 and higher), the procedures generated by **snns2idl** should be combined with the driver procedure as described in the **fluxnet/source/README** file and in **main.pro**.

## BUILDING THE EXECUTABLE

The following example is for compiling **fluxnet** (or **fluxnet.exe**). The procedure is the same for **fluxnetp** (using **netp**). Compile the main program **fluxnet.c** and the network file **net.c** (generated by SNNS):

| | |
|---|---|
| **cc -O fluxnet.c net.c -lm -o fluxnet** | **(basic Unix; use “gcc” for Linux)** |
| **bcc fluxnet.c net.c** | **(Borland C; use “cl” for Visual C)** |

You should now have the executable file **fluxnet** or **fluxnet.exe**.

# 8 REFERENCES


Ebert, E. E. and Curry, J. A., 1992, A parameterization of ice cloud optical properties for climate models, *J. Geophys. Res.*, 97(D4), 3831-3836.

Hu, Y. X. and Stamnes, K., 1992. An accurate parametrization of the radiative properties of water clouds suitable for use in climate models, *J. Climate*, 6(4), 728-724.

Key, J., 1999, Streamer User's Guide, Technical Report 96-01, Department of Geography, Boston University, 94 pp.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986, Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. 1, D. E. Rumelhart and J. L. McClelland (Eds.), Cambridge, MA: MIT PRess, 318 – 362.

Stamnes, K., Tsay, S.-C., Wiscombe, W., and Jayaweera, K., 1988, Numerically stable algorithm for discrete-ordinate-method radiative transfer in multiple scattering and emitting layered media, *Appl. Opt.*, 27, 2502-2509.

Toon, O. B., McKay, C. P., and Ackerman, T. P., 1989, Rapid calculation of radiative heating rates and photodissociation rates in inhomogeneous multiple scattering atmospheres, *J. Geophys. Res.*, 94(D13), 16287-16301.

Tsay, S.-C., Stamnes, K., and Jayaweera, K., 1989, Radiative energy budget in the cloudy and hazy Arctic, *J. Atmos. Sci.*, 46, 1002-1018.

# 9 REVISION HISTORY

The following table gives a brief description of the changes in each major and minor release of *FluxNet*.

| Version | Date | Modifications |
|---|---|---|
| 4.0 | October 1, 2001 | Changed profile levels from pressure to sigma levels; changed cloud pressure input to cloud temperature; added many colder and higher elevation polar profiles to training; trained with Streamer v3.0. |
| 3.1 | 1 June 2001 | Changed input profile levels from pressure to sigma levels, so input files that use profiles are not compatible with previous versions; retrained the network with Streamer v3.0; added a normal random number generator for better sampling of some variables in training; added another sample application. |
| 3.0 | 15 July 1999 | Added TOA fluxes to output; added surface type to input; expanded ranges of some of variables. |
| 2.1.1 | 5 April 1999 | Modified manual to reflect “bug” in specification of cloud water content. Important! |
| 2.1 | 9 Sept 1997 | Retrained with greater number of dark and clear sky cases; streamlined **run.f**. |
| 2.01 | 22 May 1997 | Fixed bug concerning incorrect cloud top temperature in Streamer run. |
| 2.0 | 20 April 1997 | One network outputs shorwave and longwave fluxes; redesigned network for use with global TOVS profiles; simplified steps for redesign and implementation. |
| 1.0 | 28 Aug 1996 | Version 1.0 release. Expanded the albedo range for snow; revised the documentation. |
| 1.0B | 16 July 1996 | First Beta test release. |