

ROOT_JETNET Interface

**Chris Neu, Catalin Ciobanu, Phil Koehn,
Richard Hughes, Brian Winer**
The Ohio State University



**Workshop on Advanced Multivariate
And Statistical Techniques**

May 31, 2002

- Motivation
- ROOT_JETNET interface
- Examples
- Summary



Motivation

- **JETNET** is a **NN package** written in **FORTRAN**:
 - Facilitates training/testing of networks
 - Anonymous ftp from: thep.lu.se (latest version 3.5)
 - Widely used and well understood
 - *Very useful*
- **ROOT**: “an object oriented framework aimed at handling the data analysis challenges in HEP”:
 - Aids users in managing large amounts of data
 - Widely used and well understood
 - *Very useful*



ROOT_JETNET Overview

- **Goals:**

- Establish a flexible and simple way of running JETNET from ROOT
- Allow non-FORTRAN programmers access to JETNET
- Preserve JETNET functionality

- **Where can I obtain the `Root_Jetnet` package?**

- **CDF members:** offline CVS repository, package name = `Root_Jetnet`
- **Others:** http://cdfpc2.mps.ohio-state.edu/root_to_jetnet/rtj.html

- **Documentation:**

- CDF Note 5434, “A ROOT Interface to JETNET”
- Included in `Root_Jetnet` code package
- Also useful: ROOT manual, JETNET 3.5 manual (see the Note for cites)



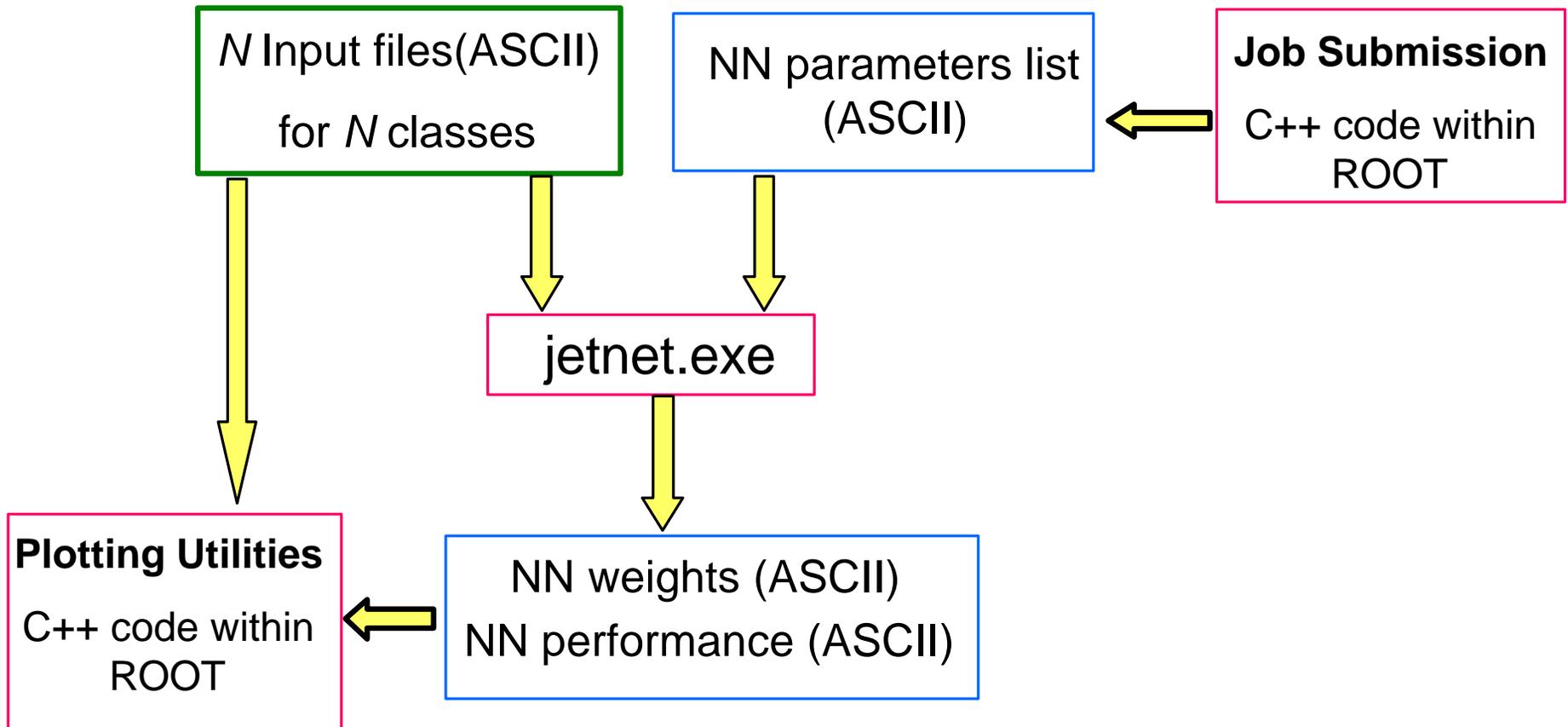
ROOT_JETNET Overview

The ROOT_JETNET interface has **three components**:

- **ROOT-side C++ class definition and scripts:**
 - **Choose and set the NN parameters**
 - **Run the NN learning job**
 - **Plot** input variables, NN performance, error, and output distribution
 - Can be modified by user according to needs
- **JETNET.exe (FORTRAN):**
 - **Input:** Data files for each class of input and NN parameter file
 - **Performs training/testing** with JETNET subroutines
 - **Output:** Performance and weights file
 - User will not need to change it (typically) but can do so if needed
- **ASCII files:**
 - **Data files** for different classes
 - **NN parameter file**
 - **Weights** files from trained net, **performance** file documenting learning progression.



ROOT_JETNET Overview





ROOT_JETNET Overview

- Changes since Run 2 AAG Workshop (2/2001):
 - Weights can be initialized to non-random values
 - Now usable on **IRIX** (formerly just LINUX)
 - **New plotting options**
 - **Access to more JETNET tunable parameters**
 - **Root_Jetnet class definition:**
 - **Formerly:** *one cumbersome script* (`root_to_jetnet.C`)
 - **Now:** **RJ class** defined by `Root_Jetnet.hh, .cc`; these methods are called via **user-written scripts** such as `Root_Jetnet_macro.C`
 - **Updated documentation**
- Documentation describes all methods and how they can be used
- Best way to really learn how the Root_Jetnet package can be used is to go through a brief example
 - Sorry...Laptop projection not good for this room, otherwise we could have an interactive introduction!



Using the ROOT_JETNET Interface: A First Example

As an instructive exercise, let us consider the task of discriminating two classes of simulated physics events. We seek to train a network that discriminates between **ttbar and W+jets background events**:

Note: *up to 10 different classes* can be presently accommodated in Root_Jetnet (more can be added).

Note²: this example (including sample input files) is included in Root_Jetnet package.

Steps:

-1. Obtain and compile the FORTRAN in Linux and C++ class in ROOT.

0. Input Files:

- ASCII; constructed by user
- Best to include a quasi-global set of **~all possible inputs** one may choose from (this is a convenience issue); *up to 20 possible inputs* can presently be accommodated in Root_Jetnet (more can be added)
- **1 file per sample** (signal, bkg1, bkg2, ...)
- **Rows** (1 row/event) and **Columns** (1 column/possible input dimension)



Input File Format

Input space consists of the following 9 variables:

- E_T jet 1,2,3
- h jet 1,2,3
- Missing E_T
- H_T
- Number of jets with $E_T > 10 \text{ GeV}$, and $|h| < 2.75$

```
Et jet 1
Et jet 2
Et jet 3
Eta 1
Eta 2
Eta 3
Met
Ht
Njet
54.65887451 40.88462067 23.49944496 -1.06890965 -1.96102273 -0.79334229 49.09893036...
36.25290298 30.39608765 28.32822800 -1.17113495 -1.06486785 -1.40596402 32.69572830...
140.77398682 28.88099861 21.95326996 -0.18924452 -0.79315042 -0.07041404 124.64159393...
96.64276123 68.13753510 60.78130341 0.52958304 0.03265952 -1.33528435 43.20137024...
43.22183609 35.94001389 25.91762161 0.55198044 -0.49529329 -0.27864811 55.30059433...
151.61328125 69.37616730 55.84277344 -0.56662613 -0.58165526 -0.06336756 41.51983643...
81.05252075 26.22723579 17.36202240 -0.44931754 0.78753597 -0.34974739 42.99569321...
103.23712158 75.03862000 60.64249420 -0.51652193 1.77900207 -1.20707572 62.73186111...
65.72234344 18.69539452 15.88183308 -0.41062140 1.20873225 0.31330562 36.68163300...
```



Setting the NN Parameters

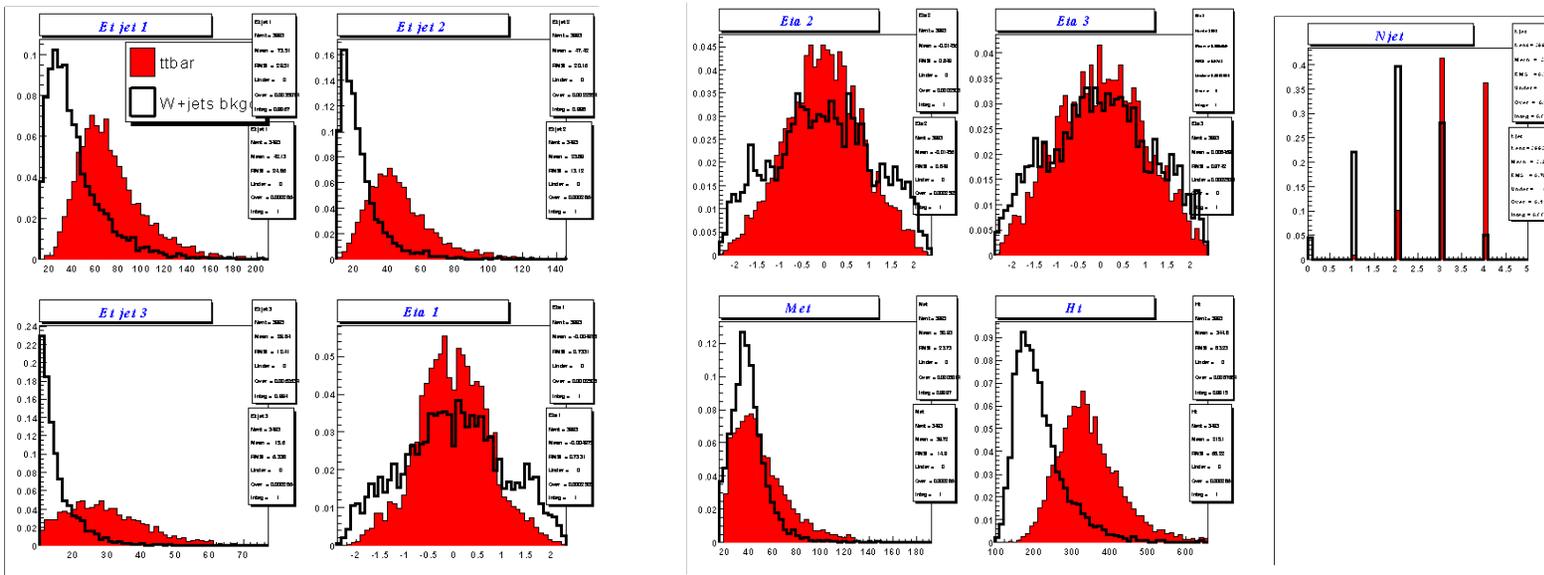
1. Within ROOT session, the user can now set various parameters that are relevant for the task at hand:
 - Define the **input data files** (with sufficient pathnames included)
 - **Direct the output** of the JETNET exe to some **user-defined directory**
 - Define the desired **NN architecture**. *We choose 9-9-1 here.*
 - Number of **learning epochs**. *We choose 1000 epochs here.*
 - **Minimization technique**.
 - **Targets for each output node for each input class:**
 - We have two input classes, and one output node.
 - It is natural then in this binary situation to demand $\text{NNOut}(\text{signal}) = 1$ and $\text{NNOut}(\text{bkgd}) = 0$
 - **Various other settings** like update frequency, momentum...

For all these tasks one uses descriptively-named methods such as “setEpoch(1000)”. See the documentation for details.



Input Variables to NN

- User may plot the variables for input samples:
 - There are **two classes**, **nine input dimensions**.
 - Red** is signal, **Open** is background.
 - Use method `plotInput1D()`





Train the NN

3. One must choose which of the 9 inputs we will use:
 - For this example, *let's use all nine*.
 - This is represented in the code by a nine-character TString “111111111”
 - If instead we wanted to forsake one of the inputs and *train on only 8 of the input dimensions*, we would use “111011111”
 - In this way we can *easily switch between different input configurations*

4. And one must now create the parameters file:
 - Default name: **netspecs.dat**
 - Created with method `netfile()`
 - This ASCII file documents all the user-desired settings applied above
 - **netspecs.dat** is one input to **jetnet.exe**



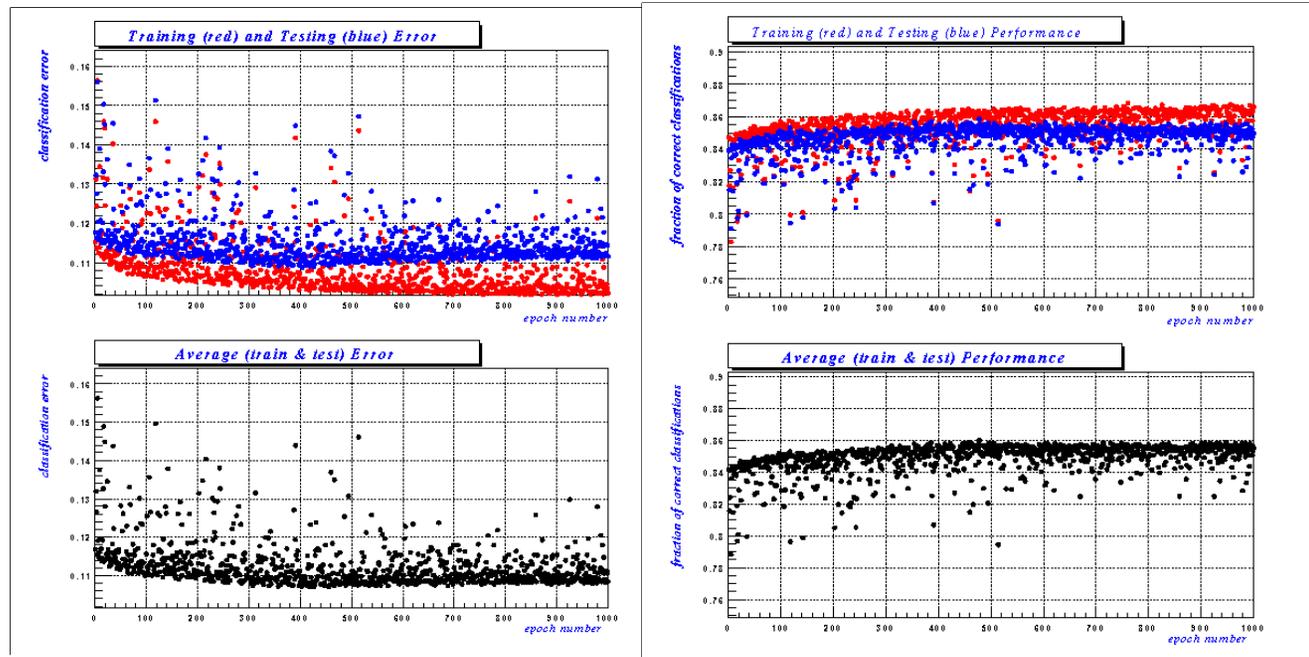
Train the NN

5. **Train the network:** Run in default mode, using current parameters:
 - Executed via method `jetnet()`
 - JETNET exe is called from within ROOT via `gSystem->Exec(...)`
 - **Inputs are scaled** to approximately the same range before training
 - NN is trained according to parameters in `netspecs.dat`
 - Output is a file containing NN weights for trained network
 - A piece of C code is generated that can be cut and paste into a macro
 - This code can be used to process new events.
6. **Examine performance** of NN learning and NN **output**



Checking Results: Performance and Error

- **Mean square error** and **performance** values per epoch number.
- *Performance is an arbitrary benchmark and should not be relied on for making judgments on NN adequacy.*
- Recall that the network is “trained” by **iterative adjustment of weights** in order to **minimize the mean square error**.
- Watch for *overtraining*



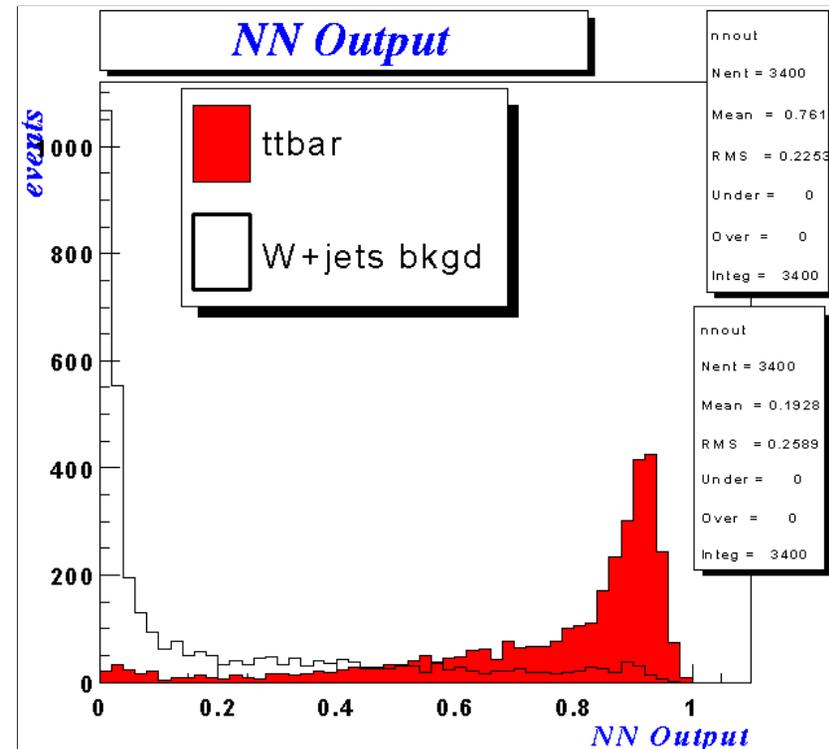
Plots made with
method
`plotPerform()`



Checking Results: NN Output

NN Output:

- Plot made with method `plotNN()`
- **Good separation**
- One can be confident that NN training was successful
- Note that when gauging the performance of any NN, one should use **a data sample that is independent** from the sample used to train the net.





Other Options

There are a variety of ways one can train networks within Root_Jetnet:

- Run in a **single shot mode**, using current parameters (above).
 - Method: `jetnet()`
- Choose **different input variables** and run again, all other parameters remaining the same.
 - Method: `jetnet("110100111")`
- **Loop over numbers of hidden nodes.**
 - Method: `jetnet("110100111", 9, 18)`
 - Generates 10 performance and 10 weights files, one set for each configuration
- **Iteratively train NNs for every possible input configuration** for various numbers of hidden nodes
 - Method: `jetnet(Int_t IterativeStudyFlag)`
 - Generates MANY files and takes a long time.



Summary

- We have a flexible interface which allows running the JETNET executable from ROOT. The interface allows the user to:
 - Easily make modifications to architecture and input set
 - Run in different modes (single, loop over inputs, loop over hidden nodes..)
 - Make various plots (performance, error, output)
 - Produce the NN function in a convenient format (.C function)
- Possible improvements:
 - Online comparison of Bayes discriminant to NN output in input space for gauging learning adequacy
 - Allow input from ROOT files
 - Vary learning parameters in training (rate, momentum) in the same way the number of hidden nodes is varied
 - Implement additional hidden layers
- Questions/Compliments/Complaints:
 - Email: neu@fnal.gov, pkoehn@fnal.gov, catutza@fnal.gov