Original

Schiller, H.:

Neuronal Network for Simulation of an Inverse Model.

In: 3rd Bio-Optical Algorithm and Protocols Workshop. Greenbelt (USA), 12.05.1994 - 13.05.1994, 1994.

Neuronal Network for Simulation of an Inverse Model

H. Schiller

May '94

Inverse modeling of the influence of CASE II water properties on the radiances measured by CZCS is known to give good results. As inverse modeling is quite a heavy computational load we are interested in methods which make feasible the inverse modeling operationally.

1. Inverting the Model
   Suppose there is a model $\mathcal{F}$ which, using some parameters $\vec{p}$, derives from concentrations of Gelbstoff etc, $\vec{c}$ the radiances $\vec{r}$ seen by a satellite in different channels:

   $$\vec{r} = \mathcal{F}_{\vec{p}}(\vec{c})$$

   Then what one needs is the inverse model

   $$\vec{c} = \mathcal{F}_{\vec{p}}{}^{-1}(\vec{r}^{\ meas.})$$

   A common procedure to realize the inverse model from the direct one is the least square method. That means one iterates the c's until

   $$\chi^2(\vec{c}) = \sum_{rad.}(\frac{r_i^{meas.} - r_i(\mathcal{F}(\vec{c}))}{\Delta r_i^{meas.}})^2 \stackrel{!}{=} Minimum$$

   If the minimization is successful then the model is inverted for the given r–values:

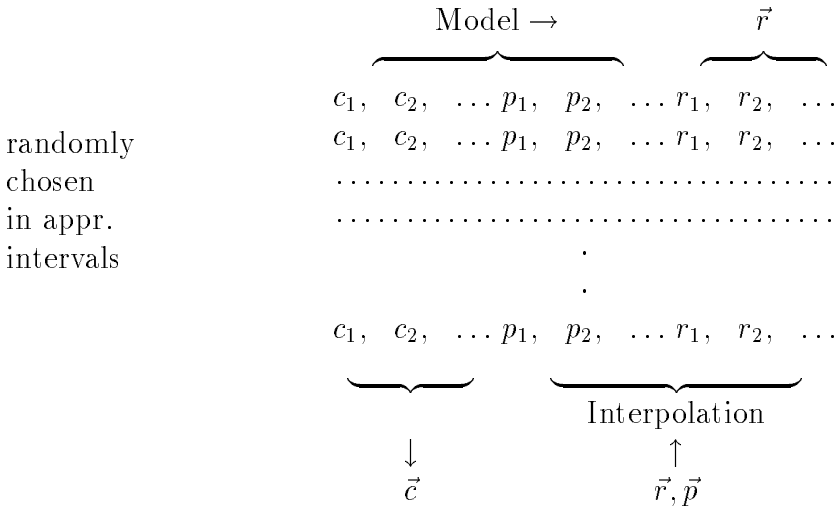
   $$\chi^2(\vec{c}) \approx 0 \quad \rightarrow \quad \chi^2_{Min.}(\vec{c}) \sim \mathcal{F}^{-1}(\vec{r})$$

   As stated above the utilization of the inverse model is quite a heavy computational task. Therefore we look for possibilities to improve the performance:

   - In Applied Optics, Vol 32, No. 18, pp 3280–3285 we demonstrated the usage of a Chebyshev–Expansion of

     $$\mathcal{F}^{-1}(\vec{r})$$

     . This method becomes clumsy if one wants to take into account additional parameters.
   - The calculation of the inverse model can be looked at as an interpolation task. Suppose we use the Model $\mathcal{F}_{\vec{p}}(\vec{c})$ to generate a huge table like



Given parameters $\vec{p}$ and measured $\vec{r}$ in principle we could use such a table to get the desired $\vec{c}$ by interpolation. Technically this will not be feasible: due to the high dimension (5 parameters, 4channels from CZCS – even worse with SeaWiFS) the table would be simply to large. So we try to use artificial neuronal nets for the interpolation task.

2. Neuronal Network (feed forward)
   There are many types of artificial neuronal networks. For our interpolation task we have chosen a feed forward (error backpropagation) network (ffNN for short). In the following the essentials of this type of network is summarized.

   ffNN are organized by layers. There is an input layer, an output layer and one or more hidden layer(s) between them. Each layer consists of neurons: the input layer has as many neurons as there are input values, the output layer has as many neurons as there are output values necessary and the hidden layer(s) need a problem–dependent number of neurons.

   Between two neighbouring layers are directed links: each neuron in one layer has a link to each neuron of the next (neighbouring) layer. Each link has a weight (w).

   Each neuron calculates its output value according to

$$o(-bias + \sum_{incominglinks} w_i x_i)$$

   where

   $bias$ is a value specific for each neuron

   w_i is the weight of the link

   x_i is the output–value of the link in the preceding[1] layer

   o is a nonlinear function. We used the (commonly used) logistic function $o(s) = \frac{1}{1+exp(s)}$.

   The ffNN works sequentially: at first the input–values are applied to the input–neurons and their outputs are calculated. Then all neurons of the first hidden layer calculate their

[1] The neurons in the input–layer have only one incoming link and x_1 then is the input value

outputs and so on until the output–layer is reached — giving the network–results for the applied input.

For a ffNN to be useful one has to 'teach' it. For this one generates two sufficiently large sets of corresponding input–output–vectors — one set is used as 'training'–sample and the other set as test–sample. During 'teaching' the values of the *bias*es of all neurons as well as the weights w of all the links are changed so as to minimize[2] an error–function

$$\sum_{'trainings-sample'} \sum_{output-layer} (o_{desired} - o_{ffNN})^2$$

After this minimization–procedure the second ('test') set is used to check if the resulting net has 'generalization'–power, i.e. to produce reasonable results also for input–values which were not 'shown' to it before.

3. Results
We used a ffNN with nine input–neurons: four radiances from CZCS and five parameters

   1. ch. 1
   2. ch. 2
   3. ch. 3
   4. ch. 4
   5. Angstrom–coeff.
   6. ozon–concentr.
   7. Θ_{sun}
   8. Θ_{view}
   9. Azimuth—difference

   and four output values:

   1. log(aerosol)
   2. log(chlorophyll)
   3. log(suspended matter)
   4. log(yellow substance)

   The training– and test–sample contained 15K points each. Two ffNN's were 'trained': the first with two hidden layers each with 80 neurons, the second with three hidden layers with 100, 50 and 10 neurons respectively.

	mean residuum	
net	training	test
80×80	0.0021	0.0031
100×50×10	0.0018	0.0042

[2] In the 'training–phase the errors are propagated against the normal processing direction: they are backpropagated from the output–layer to the input–layer

The 'generalization'–power of the 80×80–net was better, so the following results are for this net. The figures show $o_{desired}$ vs. o_{ffNN} from the test–sample. A reasonable agreement was reached.

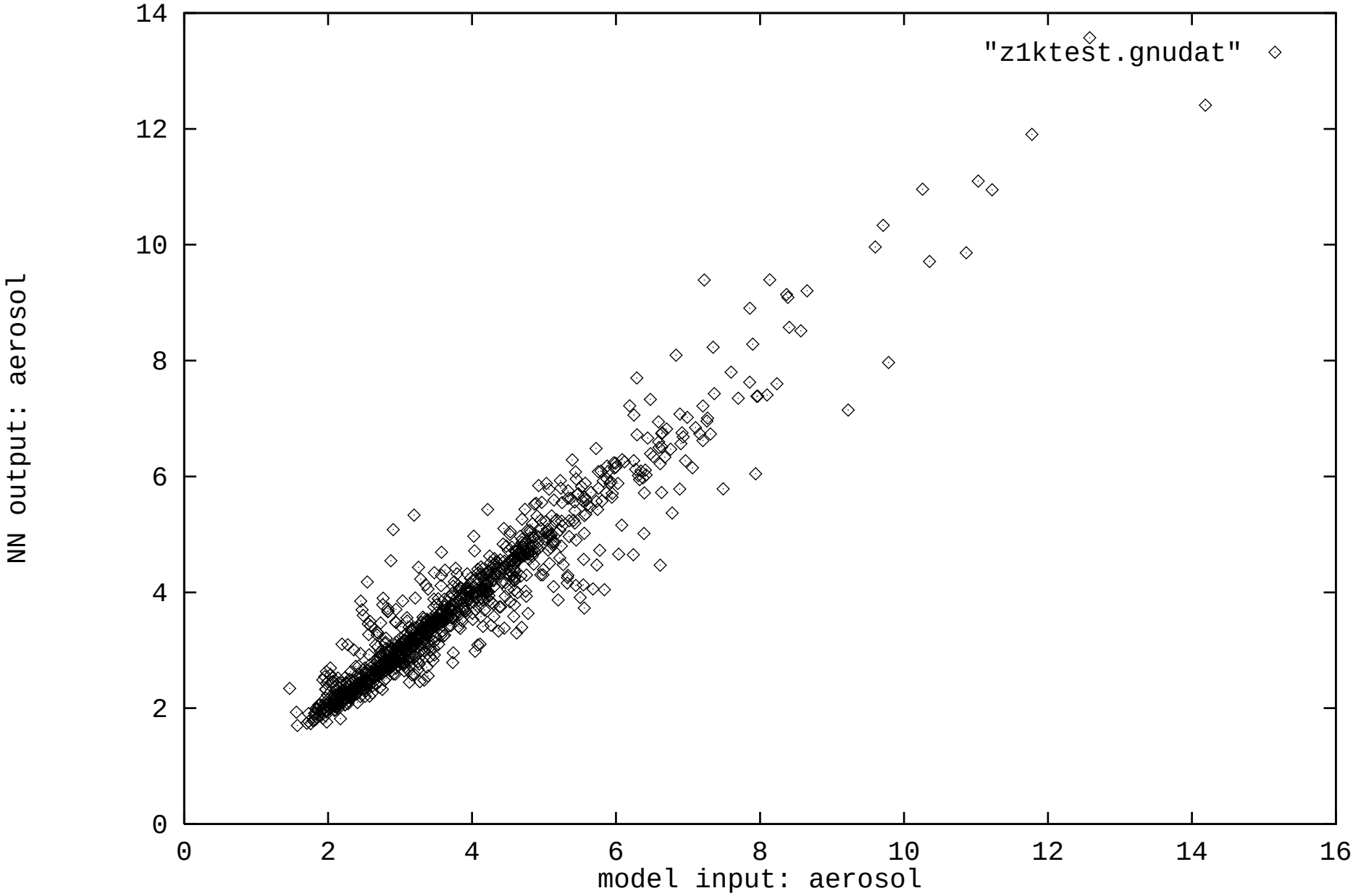
The net resulted in a data compression of $\frac{13 \times 15000}{6600} \sim 30$.

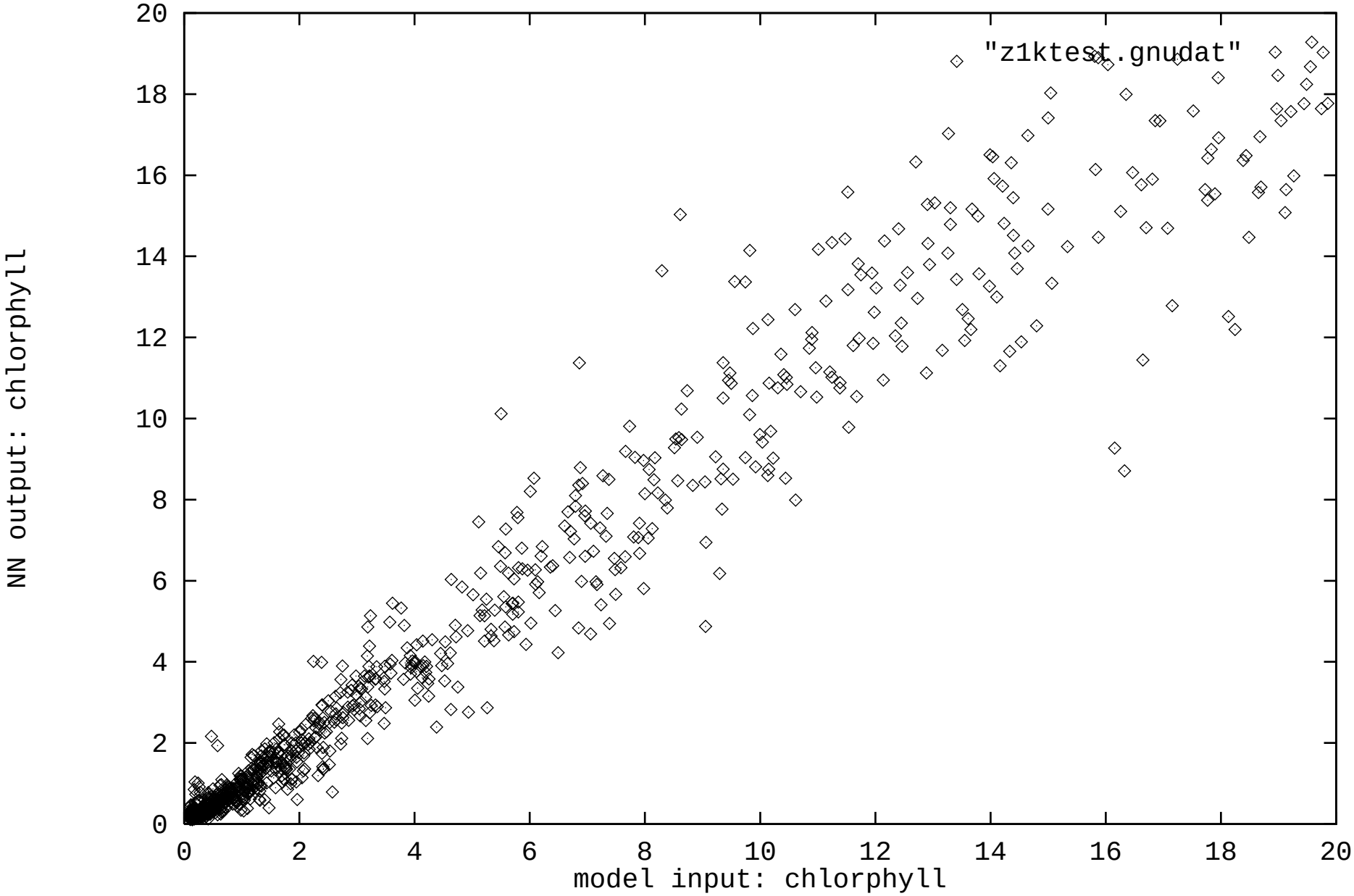
From the ffNN a C–function was generated which processed ~100 pixels/sec at a Sparc–station iPX (speed up of ~10.

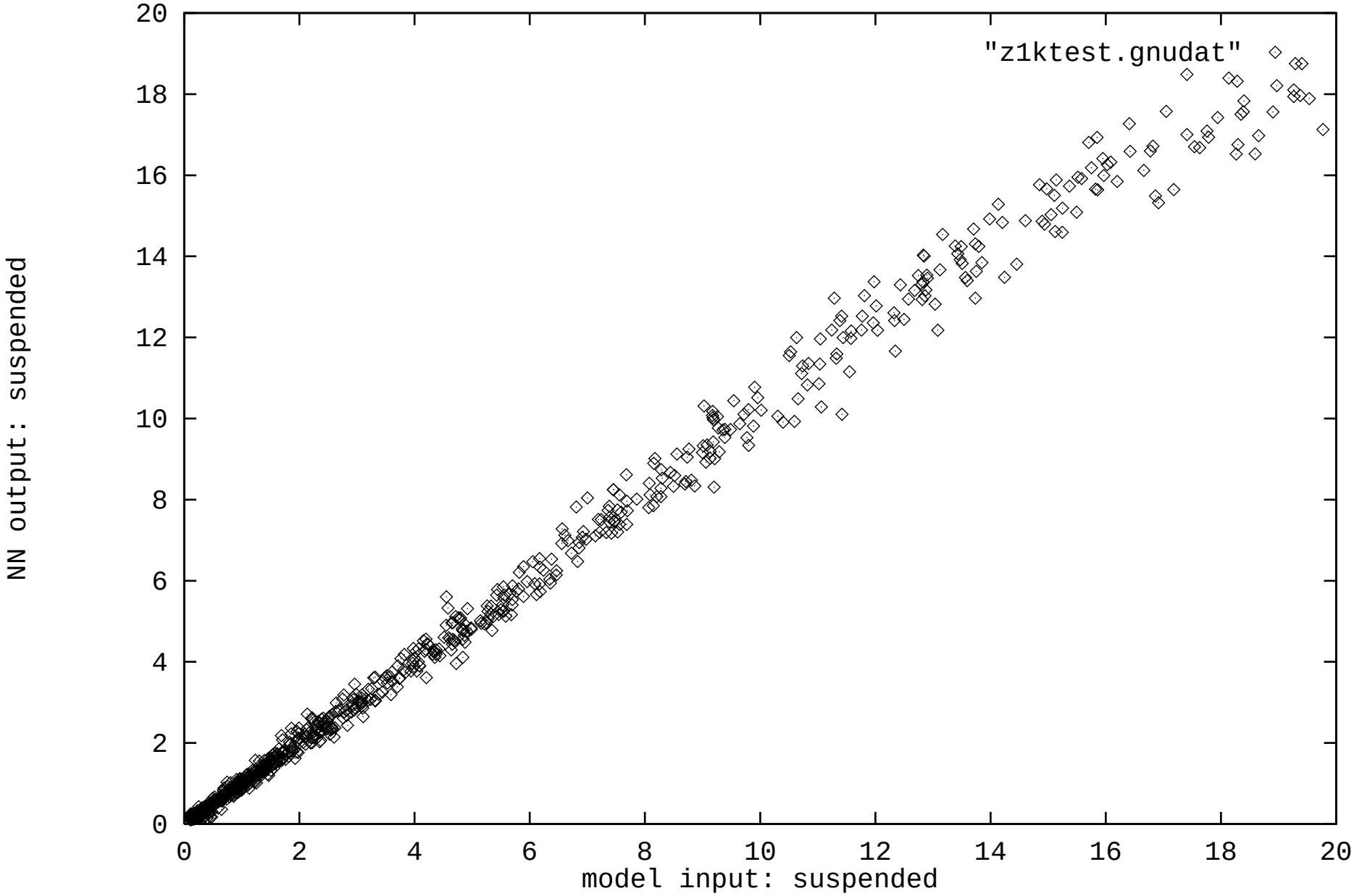
4. Outlook

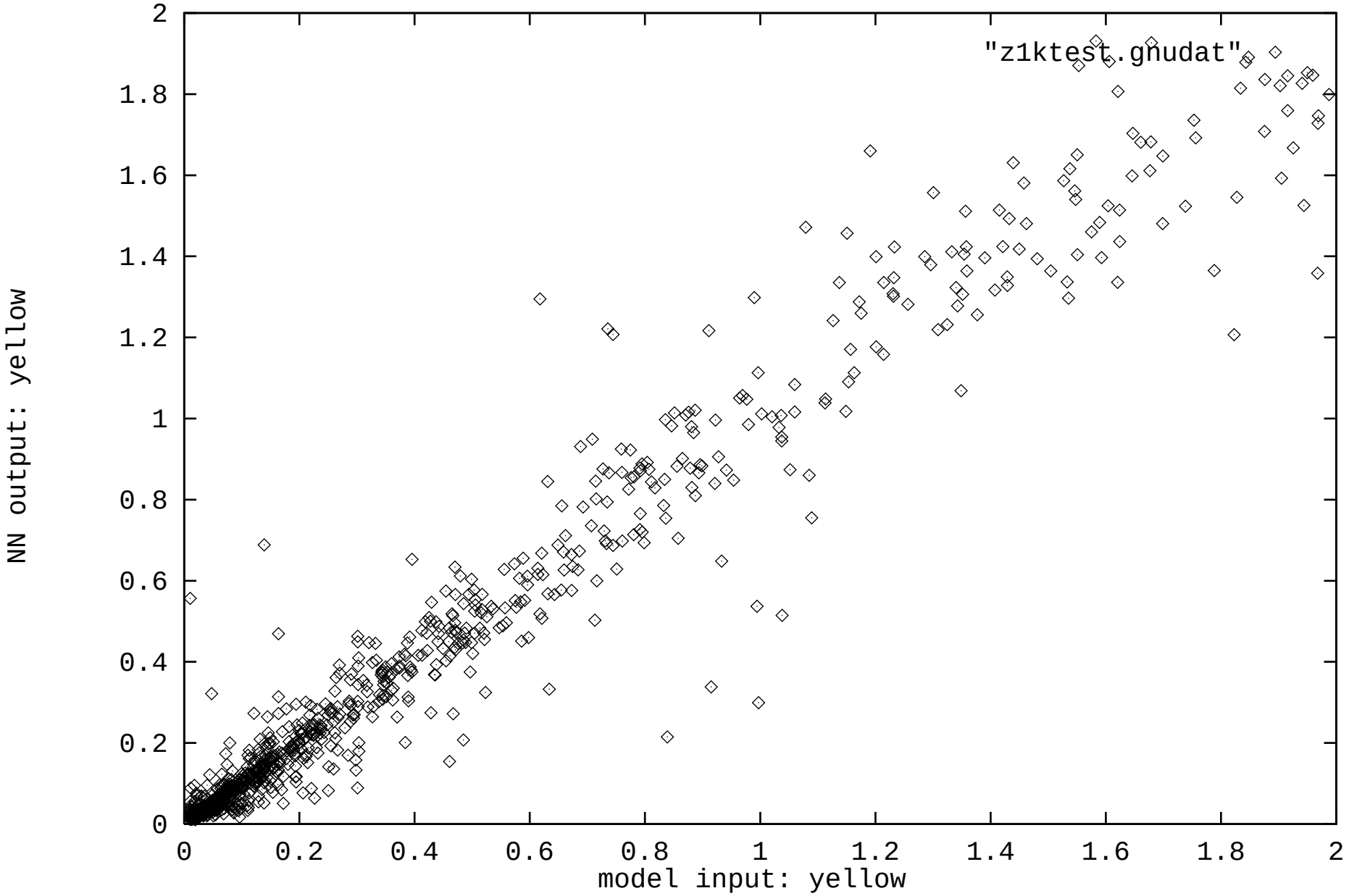
- To use χ^2 for model inversion one needs to make simplifications in order to do the inversion within a reasonable time. Such simplifications are not necessary with ffNN's.
- Operationally one could — after utilization of the ffNN — use the (direct) model $\mathcal{F}$ to check the result[3].

[3]Bad results could be used to 'reteach' the ffNN from time to time.


"z1ktest.gnudat"
NN output: aerosol
model input: aerosol
0
2
4
6
8
10
12
14
16


"z1ktest.gnudat"
NN output: chlorphyll
model input: chlorphyll
0
2
4
6
8
10
12
14
16
18
20


"z1ktest.gnudat"
NN output: suspended
model input: suspended
0
2
4
6
8
10
12
14
16
18
20


"z1ktest.gnudat"
NN output: yellow
model input: yellow
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2