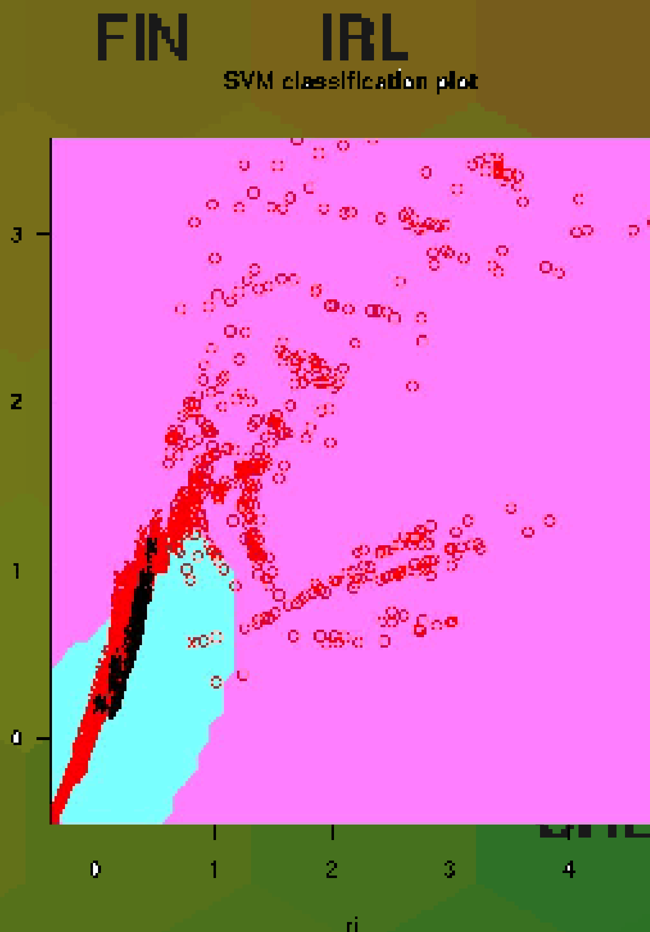
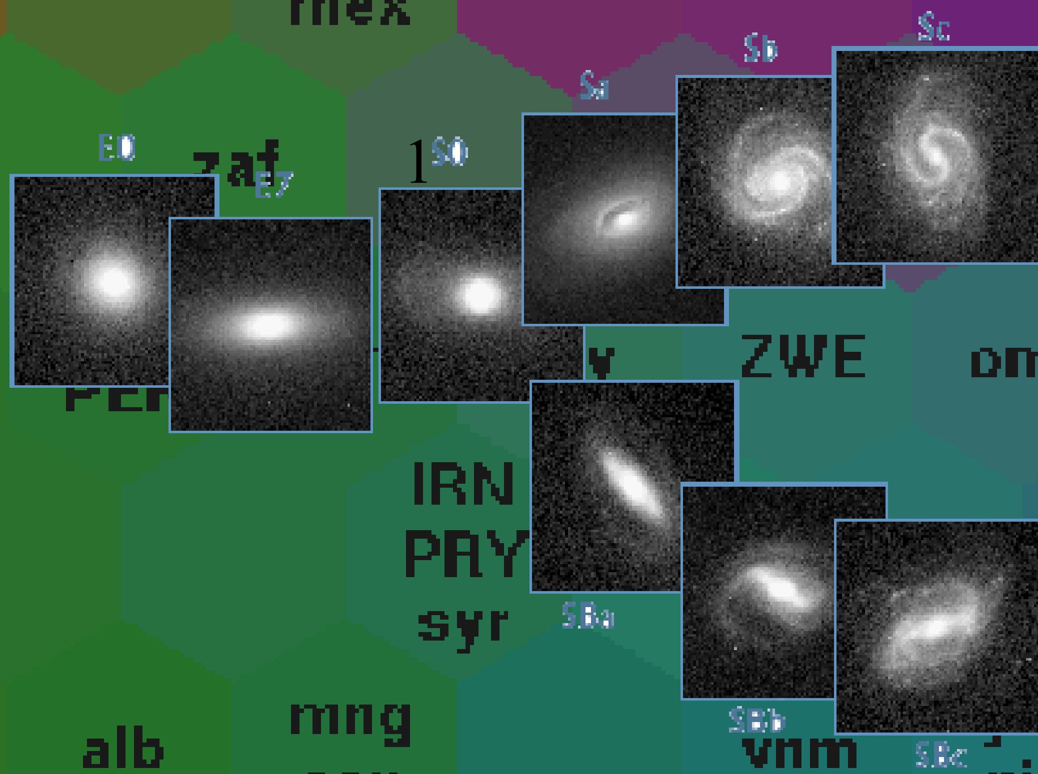
Introduction to machine learning and pattern recognition

Lecture 3

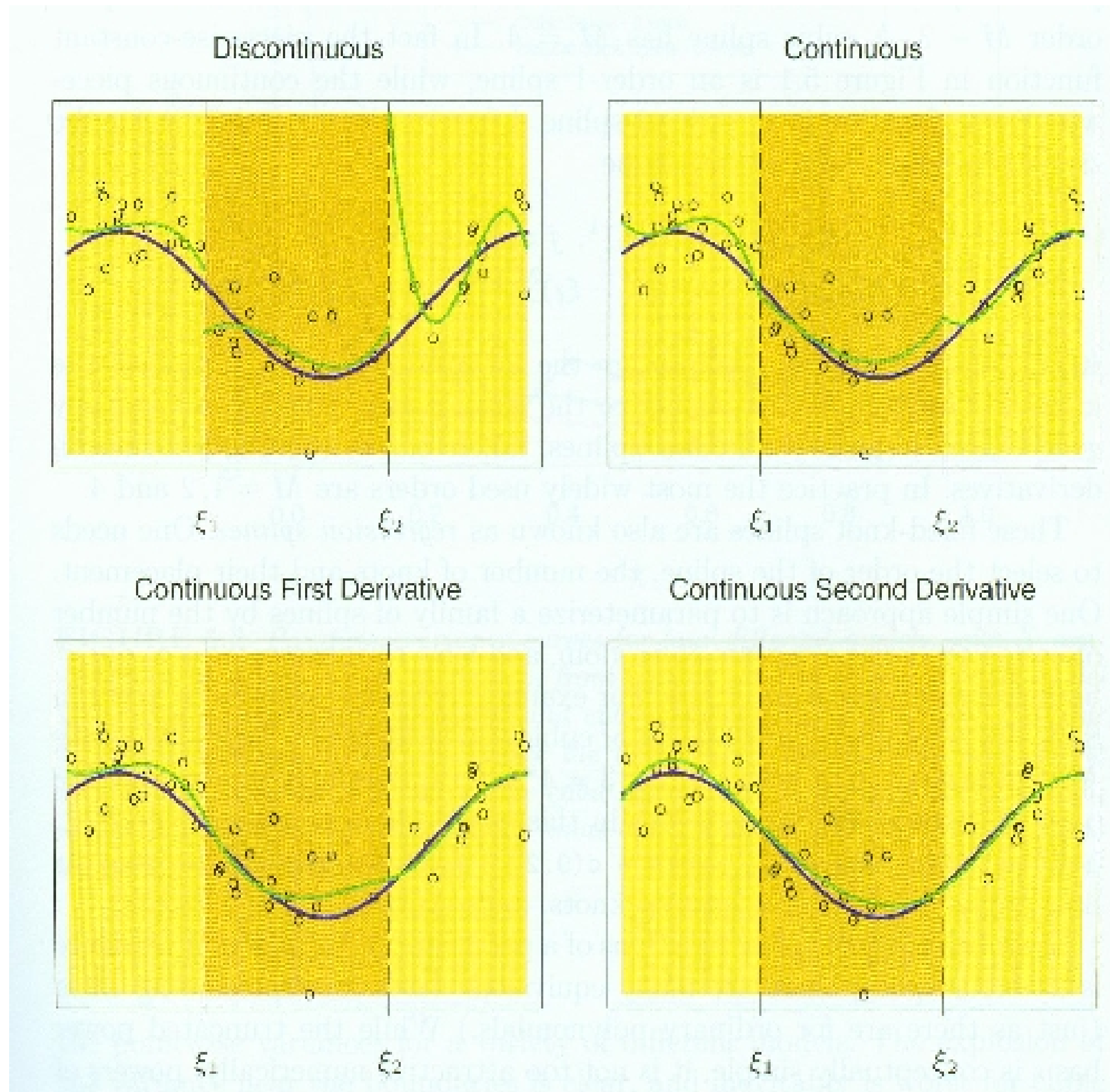
Coryn Bailer-Jones

http://www.mpia.de/homes/calj/mlpr_mpia2008.html





Piecewise cubic polynomial fits



© Hastie, Tibshirani, Friedman (2001)

Splines (one-dimensional)

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$$

- Cubic piecewise polynomial

$$h_1(X)=1,\ \ h_2(X)=X,\ \ h_3(X)=X^2,\ \ h_4(X)=X^3,$$
$$h_5(X)=(X-\xi_1)^3_{+},\ \ h_6(X)=(X-\xi_2)^3_{+},$$

  - continuous second derivatives is a cubic spline
  - continuous third derivatives is a global cubic polynomial
- cubic spline
  - has six basis function (6D projection space)
  - (3 regions) x (4 parameters per region) – (2 knots) x (3 constraints per knot) = 6
  - Generally: No. basis functions = 4 x (N+1) – 3N = N + 4, where N is no. of knots

Smoothing splines

- Avoid knot selection by using all data points as knots
  - avoid overfitting by using regularization
- Minimise a penalized sum-of-squares

$$RSS(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 \, dt$$

$f()$ is the fitting function with continuous second derivatives

$\lambda = 0 \quad \Rightarrow \quad f()$ is any function which interpolates the data (could be wild)

$\lambda = \infty \quad \Rightarrow$ straight line least squares fit (no second derivative tolerated)

Solution is a cubic spline with knots at each of the $\{x_i\}$ i.e.

$$f(x) = \sum_{j=1}^{N} h_j(x) \beta_j$$

Smoothing splines

With $f(x) = \sum_{j=1}^{N} h_j(x)\beta_j$

the residual sum of squares (error) to be minimized is

$$RSS(f, \lambda) = (\boldsymbol{y} - \boldsymbol{H}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\Omega}_N \boldsymbol{\beta}$$

where

$$\{\boldsymbol{H}\}_{ij} = h_j(x_i) \qquad \text{and} \qquad \{\boldsymbol{\Omega}_N\}_{jk} = \int h''_j(t) h''_k(t)\, dt$$

The solution is

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{H}^T\boldsymbol{H} + \lambda \boldsymbol{\Omega}_N)^{-1} \boldsymbol{H}^T \boldsymbol{y}$$

The interpolating function is

$$y(x = x_{new}) = \boldsymbol{h}(x_{new})\hat{\boldsymbol{\beta}}$$

Smoothing splines

Estimate of the function at the set of training values is

$$\hat{\boldsymbol{f}}(x) = \boldsymbol{H}\hat{\boldsymbol{\beta}}$$

$$= \boldsymbol{H}(\boldsymbol{H}^T\boldsymbol{H} + \lambda\boldsymbol{\Omega}_N)^{-1}\boldsymbol{H}^T\boldsymbol{y}$$

$$= \boldsymbol{S}_\lambda \boldsymbol{y}$$

Note that the fit is linear in $\boldsymbol{y}$ as $\boldsymbol{S}_\lambda$ depends only on x_i and λ.

$\boldsymbol{S}_\lambda$ is often called the smoother matrix.

For a linear model the effective degrees of freedom is defined as

$$df(\boldsymbol{S}_\lambda) = trace(\boldsymbol{S}_\lambda)$$

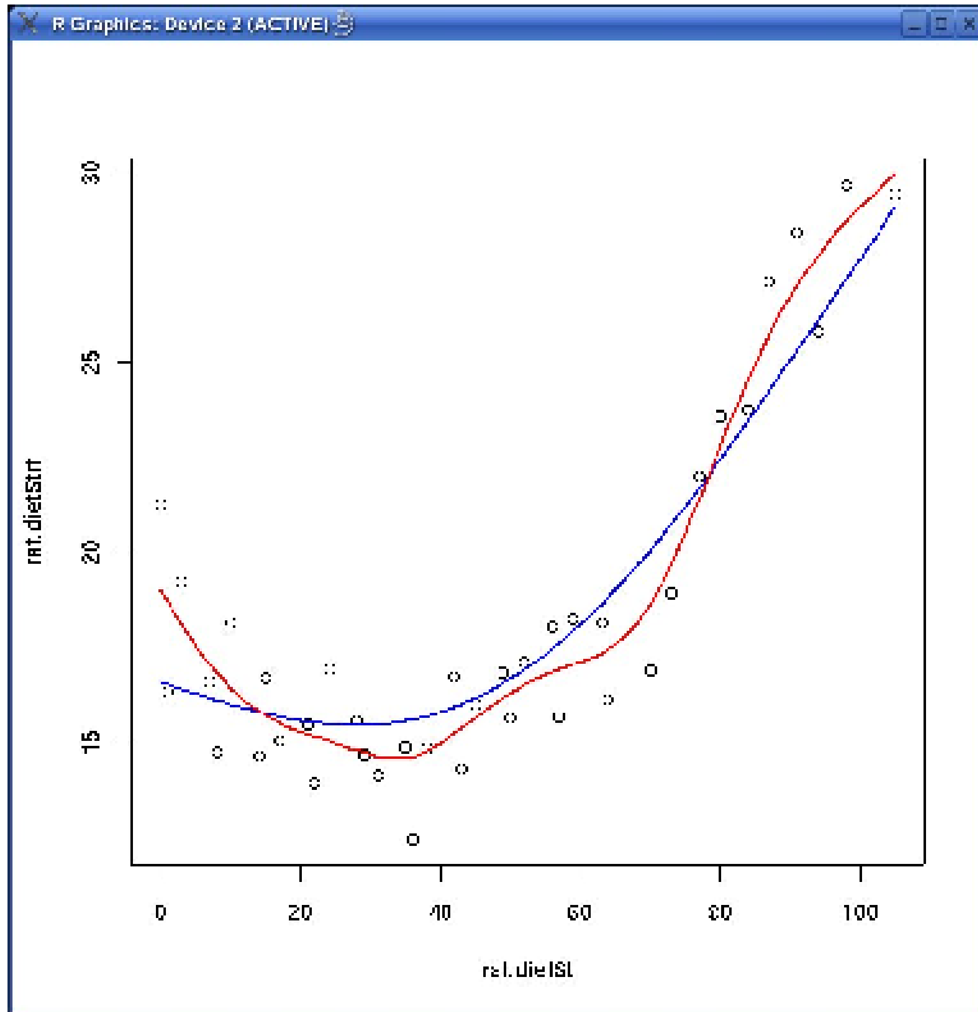
df is the dimension of the projection space = no. of basis functions = no. of parameters in the fit.

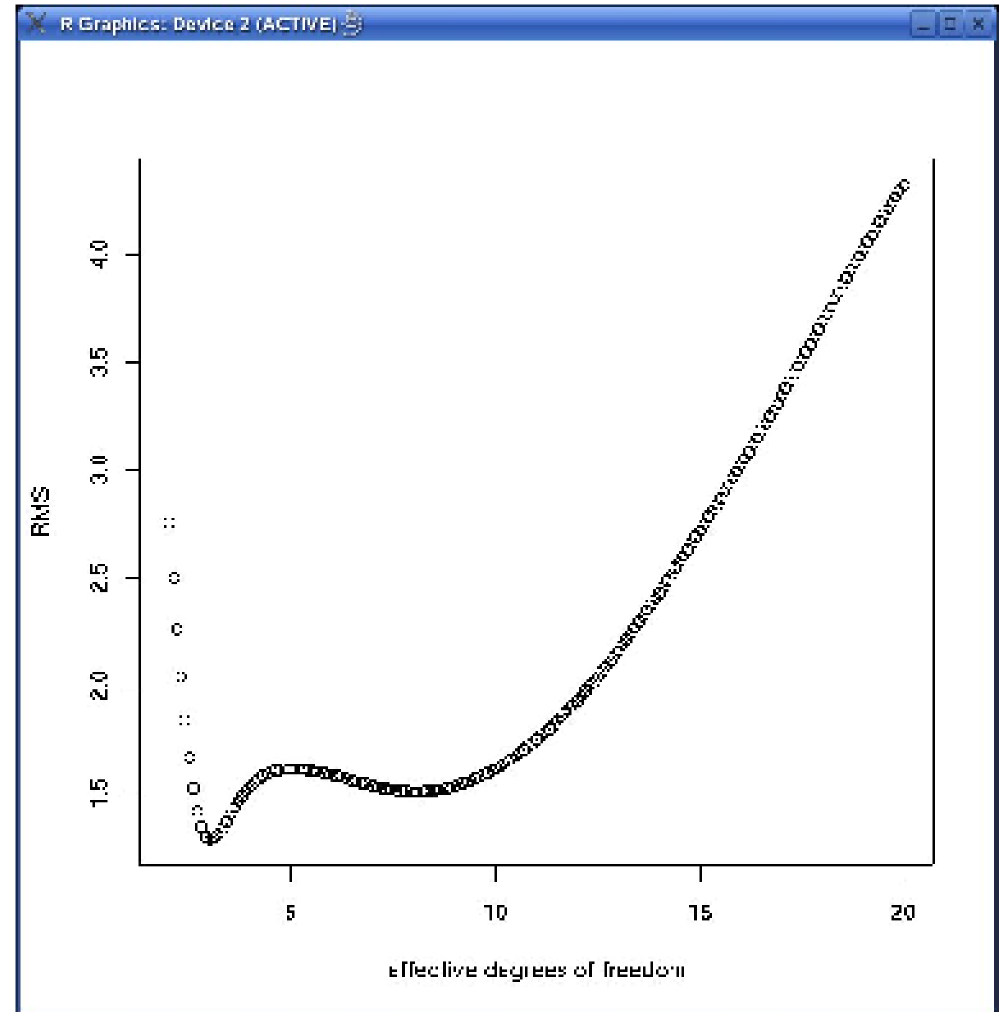
Generally more intutitive to specify regularization via df than via λ.

What is the df of a model linear in the data (x) ? Check empirically using R...
How are df and λ related?

Splines in R

See R scripts on the web





Regularization and bias-variance with smoothing splines

Properties of the smoother matrix

- it is an N x N symmetric matrix of rank N
- semi-positive definite, i.e. $\boldsymbol{x}\boldsymbol{S}_\lambda\boldsymbol{x} \geq 0 \quad \forall\, \boldsymbol{x} \neq 0$

Bias-variance decomposition

- df too low = too much smoothing
  - high bias, low variance, function “underfit”
- df too high = too little smoothing
  - low bias, high variance, function “overfit”

$$\begin{aligned} \text{Variance} &= \boldsymbol{S}_\lambda \boldsymbol{S}_\lambda^T \\ \text{Bias} &= \boldsymbol{f} - E[\hat{\boldsymbol{f}}] \\ &= \boldsymbol{f} - \boldsymbol{S}_\lambda \boldsymbol{f} \end{aligned}$$

$$\begin{aligned} \hat{\boldsymbol{f}}(x) &= \boldsymbol{S}_\lambda \boldsymbol{y} \\ df(\boldsymbol{S}_\lambda) &= trace(\boldsymbol{S}_\lambda) \end{aligned}$$

Generalized Cross Validation (GCV)

$$\hat{\boldsymbol{f}}(x) = \boldsymbol{S}\,\boldsymbol{y}$$

$\hat{f}^{-i}$ is the function estimated without point x_i. The LCV error is

$$\frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{f}^{-i}(x_i)]^2 = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i)}{1 - S_{ii}}\right]^2$$

i.e. only have to calculate once, not N times

where S_{ii} is the ith diagonal element of $\boldsymbol{S}$. The GCV approximates this

$$\text{GCV} = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i)}{1 - \text{trace}(\boldsymbol{S})/N}\right]^2$$

GCV replaces matrix elements by their mean. The trace (=df) is often easier to compute.

Basis functions in high dimensions: neural networks

Mapping with a transfer function

Problem is to fit a function to $T = T(I_{1,} I_i, \ldots, I_p)$

Let $O = \hat{T}$

$$O = f(x) \quad \text{where}$$

$$x = w_0 + \sum_{i=1}^{p} w_i I_i$$

$$= \sum_{i=1}^{p+1} w_i I_i \quad \text{by including an additional constant input (bias): set to 1}$$

$$f(x) = x \quad \text{linear model}$$

$$f(x) = \frac{1}{1+e^{-\nu x}} \quad \text{sigmoidal transfer function (fixed constant, } \nu \text{)}$$

maps an infinite input range onto a finite output range.

Variables weighted, combined and passed through the transfer function.

Going nonlinear: introduce interactions between inputs

Introduce a second stage for the transfer function

$$O = f(x) \quad \text{where} \quad x = \sum_{j=1}^{J} w_j H_j$$

for some $J \geq 1$

$$f(x) = \frac{1}{1+e^{-\nu_x x}} \quad \text{sigmoidal transfer function}$$

$$H_j = g(y_j) \quad \text{where} \quad y_j = \sum_i w_{i,j} I_i$$

$$g(y_j) = \frac{1}{1+e^{-\nu_y y_j}} \quad \text{another sigmoidal transfer function}$$

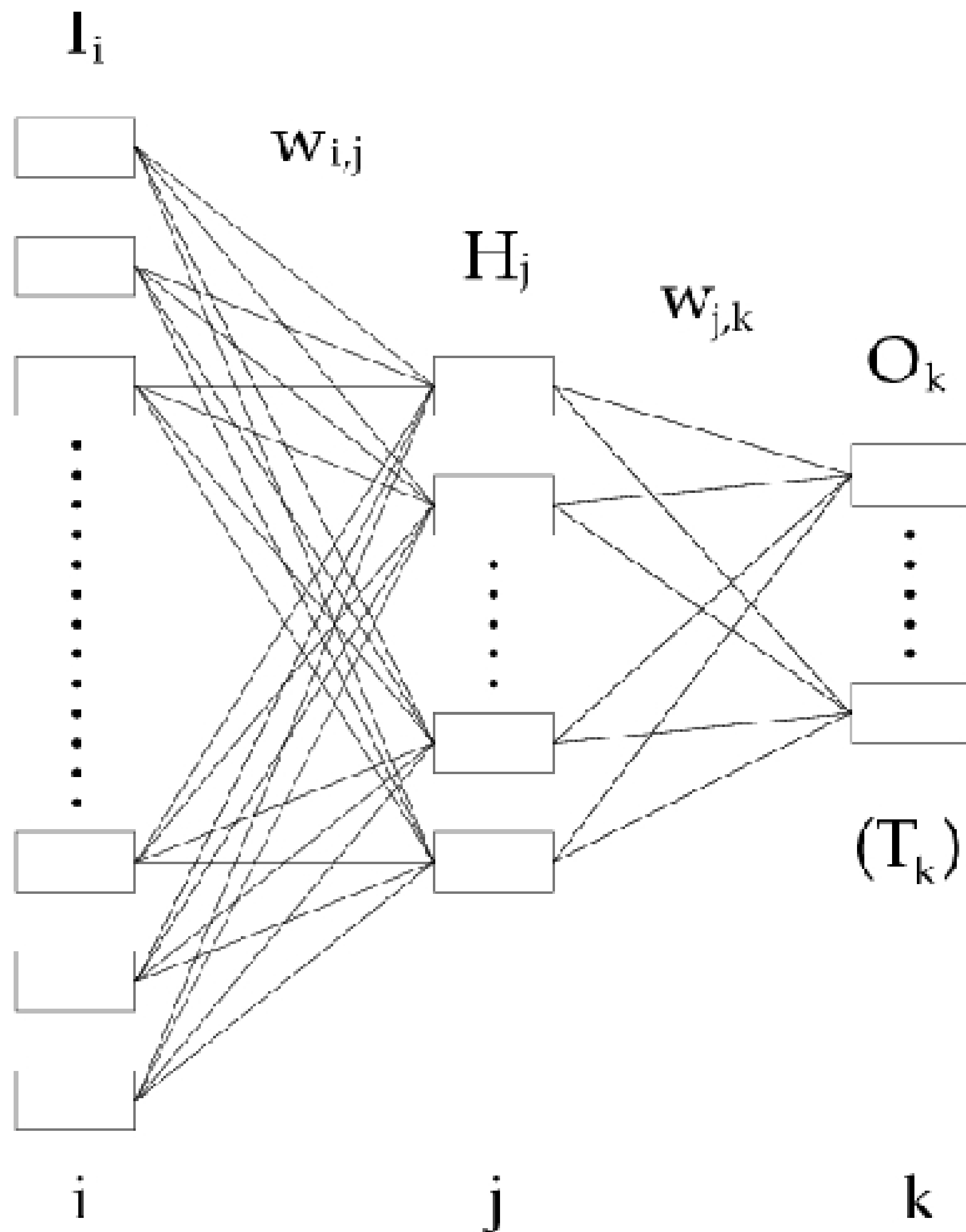
Putting it all together

$$O = \frac{1}{1+\exp\left(-\nu_x \sum_j w_j H_j\right)} \quad \text{where}$$

$$H_j = \frac{1}{1+\exp\left(-\nu_y \sum_i w_{i,j} I_i\right)}$$

This *second layer* provides J nonlinear combinations of the inputs.

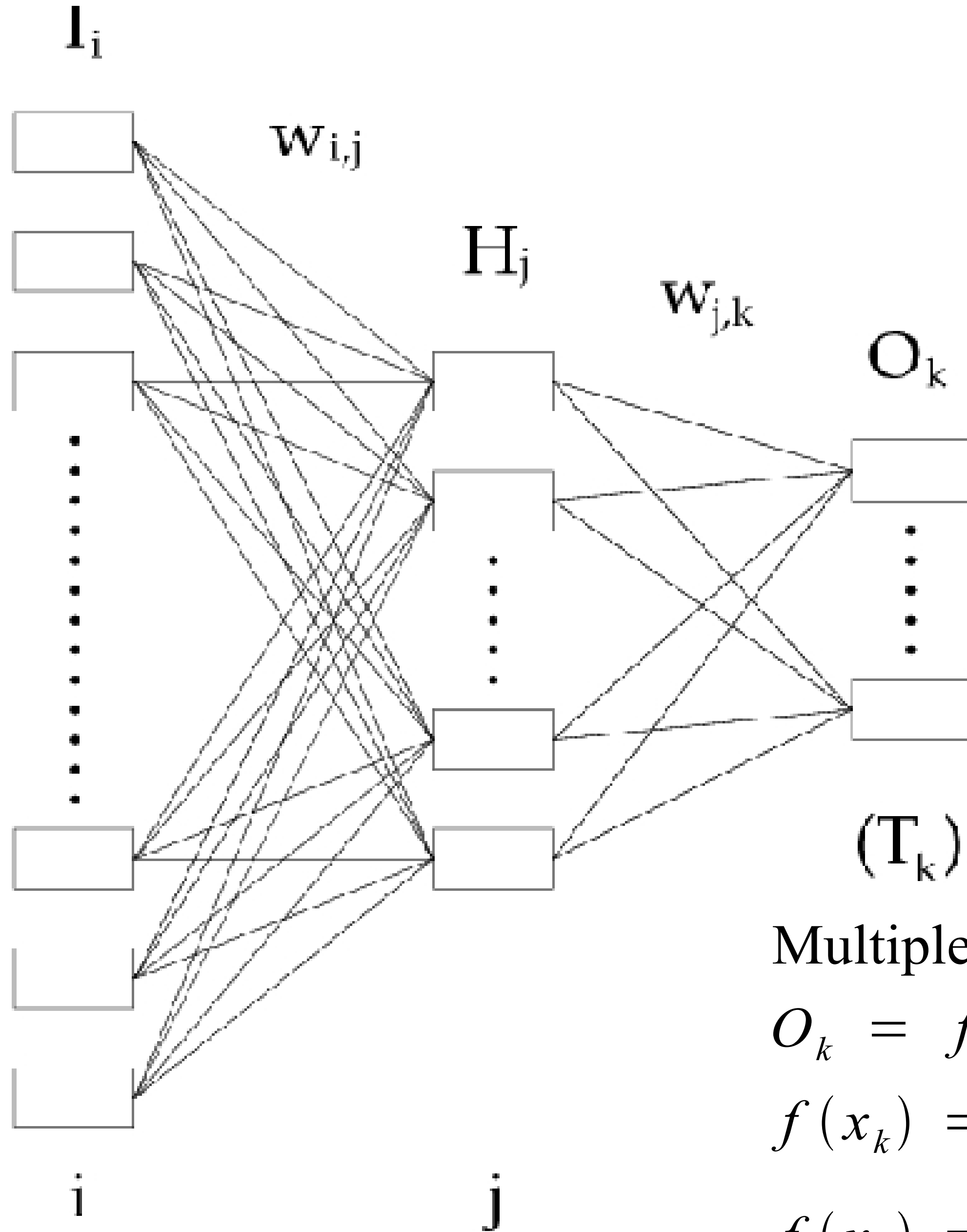
Pictogram: multilayer perceptron (neural network)



$$H_j = \frac{1}{1+\exp\left(-\nu_y \sum_i w_{i,j} I_i\right)}$$

$$O = \frac{1}{1+\exp\left(-\nu_x \sum_j w_j H_j\right)}$$

Pictogram: multilayer perceptron (neural network)



$$H_j = \frac{1}{1+\exp\left(-\nu_y \sum_i w_{i,j} I_i\right)}$$

$$O = \frac{1}{1+\exp\left(-\nu_x \sum_j w_j H_j\right)}$$

Multiple outputs:

$$O_k = f(x_k) \quad \text{where} \quad x_k = \sum_j w_{j,k} I_j$$

$$f(x_k) = x_k \qquad \text{linear model}$$

$$f(x_k) = \frac{1}{1+e^{-\nu x_k}} \qquad \text{sigmoidal transfer function}$$

C.A.L. Bailer-Jones. *Machine learning and pa*

Training the network

Supervised method. Train using labelled data.

Minimize an objective function w.r.t the network parameters (weights)

$$E = \frac{1}{2} \sum_N \sum_k \left(O_{k,n} - T_{k,n} \right)^2$$

$\{T_{k,n}, I_{i,n}\}$ training data set

Recall that

$$O_k = \frac{1}{1 + \exp(-\nu_x x_k)} \qquad x_k = \sum_{j=1}^{J} w_{j,k} H_j$$

$$H_j = \frac{1}{1 + \exp(-\nu_y y_j)} \qquad y_j = \sum_{i=1}^{N} w_{i,j} I_i$$

Therefore this is *nonlinear* least squares. Must solve numerically, i.e. iteratively.

Gradient descent

Initialize weights to random values.

Update weights in direction which reduces current error:

$$\Delta \boldsymbol{w} \propto -\frac{\partial E}{\partial \boldsymbol{w}}$$

Minimization in a $(I+1)\times J+(J+1)\times K$ dimensional space.

In practice need to adjust step size and add some 'momentum'

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w} + \alpha \Delta w(t-1)$$

We can derive how the error derivative depends on the inputs and target values.

That is we can *back propagate* the errors through the network...

Back propagation: weight update equations

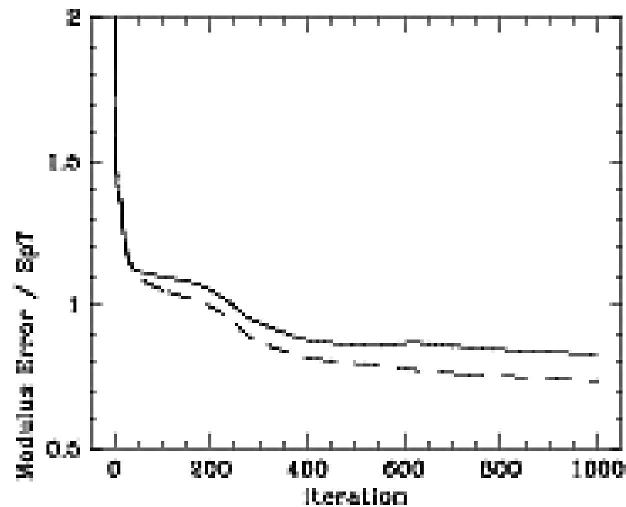
$$\frac{\partial E}{\partial w_{j,k}} = H_j (O_k - T_k) f'(x_k)$$

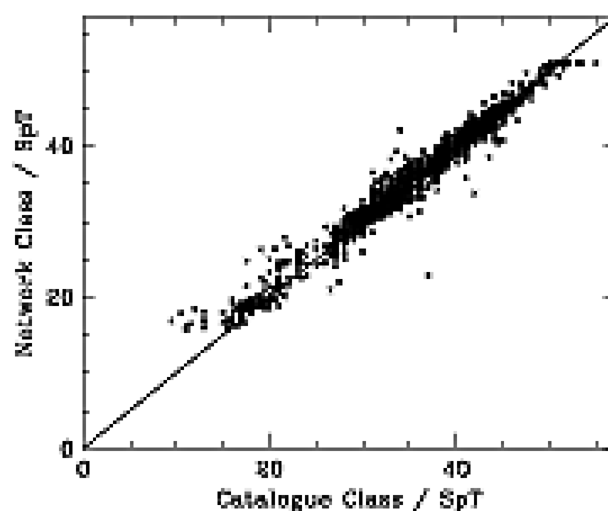
$$\frac{\partial E}{\partial w_{i,j}} = I_i g'(y_j) \sum_k (O_k - T_k) f'(x_k) w_{j,k}$$

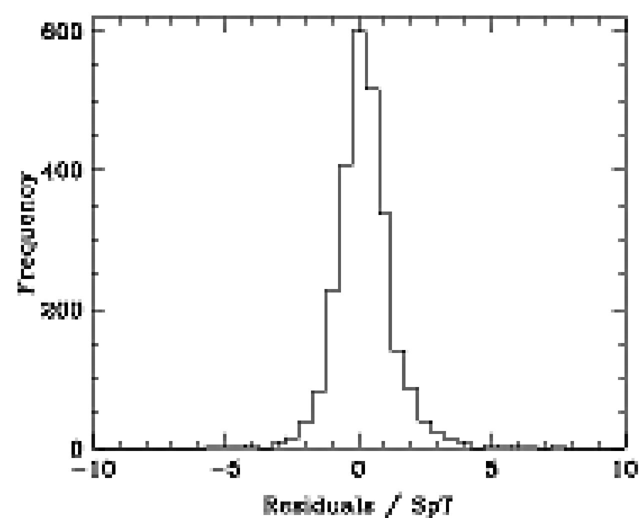
$$O_k = \frac{1}{1 + \exp\left(-\nu_x \sum_j w_{j,k} H_j\right)} \qquad H_j = \frac{1}{1 + \exp\left(-\nu_y \sum_i w_{i,j} I_i\right)}$$

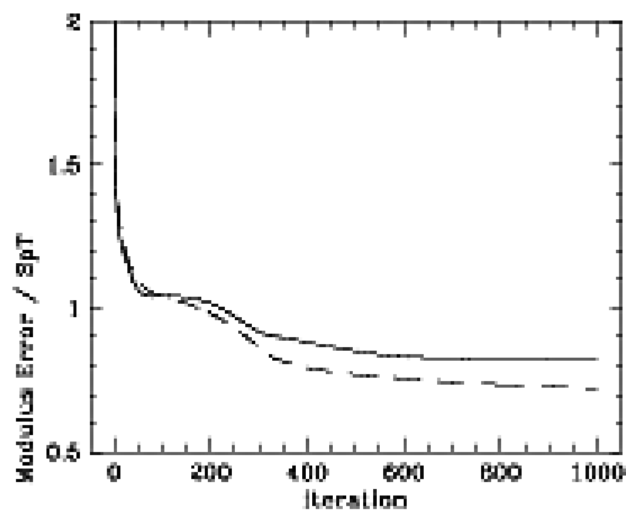
The transfer function $f(x) = \dfrac{1}{1 + e^{-\nu x}}$ has the useful property

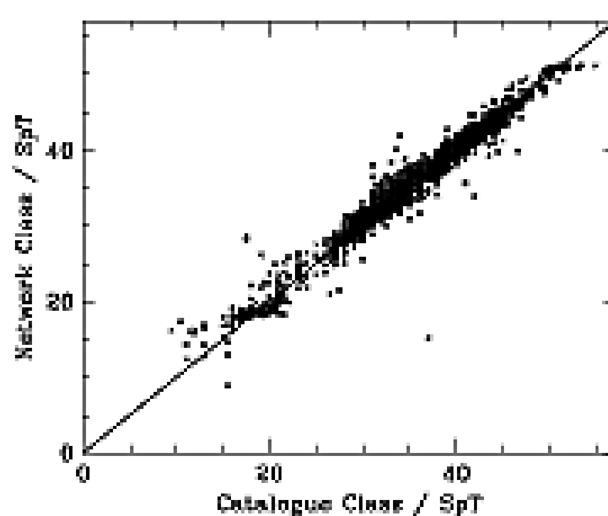
$f'(x) = \dfrac{df}{dx} = f(1 - f)$ and likewise for $g(y)$

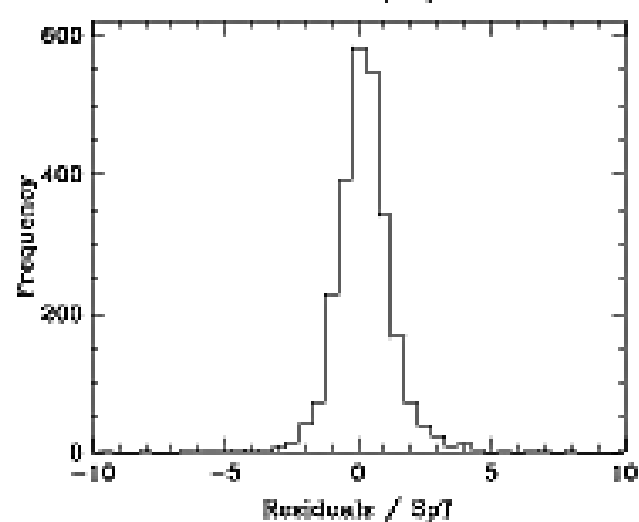

Modulus Error / SpT
Iteration
0.5
1
1.5
2
0
200
400
600
800
1000

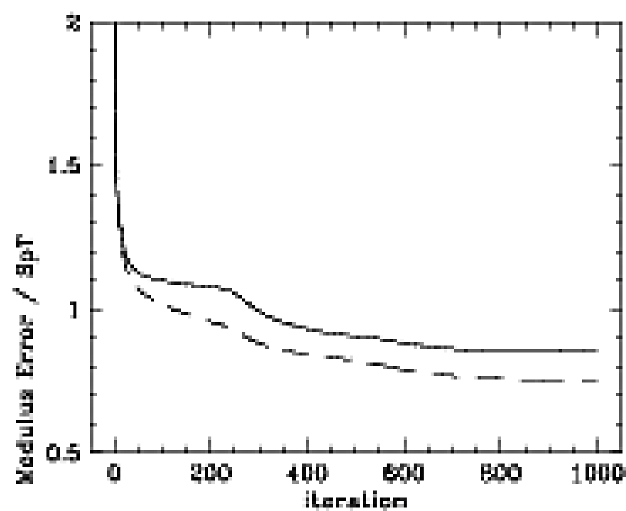

Network Class / SpT
Catalogue Class / SpT
0
20
40

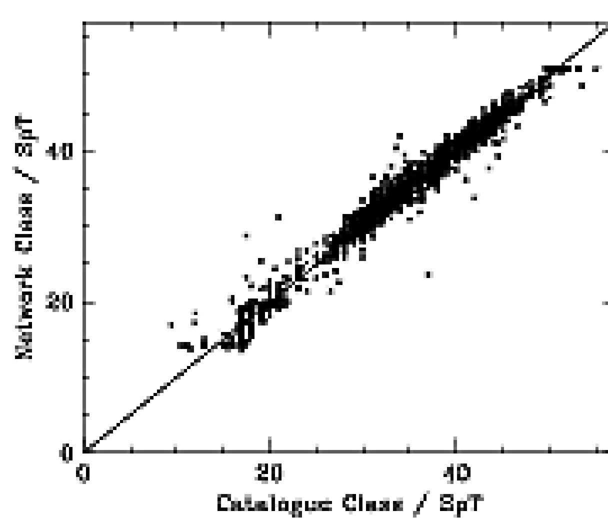

Frequency
Residuals / SpT
0
200
400
600
−10
−5
0
5
10

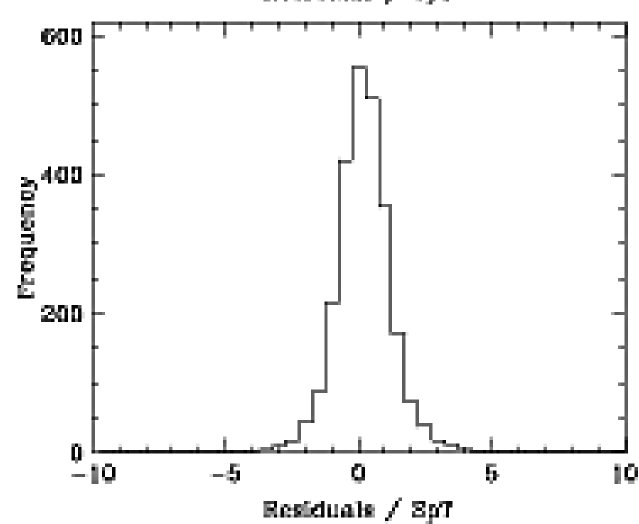

Modulus Error / SpT
Iteration
0.5
1
1.5
2
0
200
400
600
800
1000


Network Class / SpT
Catalogue Class / SpT
0
20
40


Frequency
Residuals / SpT
0
200
400
600
−10
−5
0
5
10


Modulus Error / SpT
Iteration
0.5
1
1.5
2
0
200
400
600
800
1000


Network Class / SpT
Catalogue Class / SpT
0
20
40


Frequency
Residuals / SpT
0
200
400
600
−10
−5
0
5
10

Regularization (weight decay)

Use standard regularization techniques, e.g. penalize high curvature

$$E = \frac{1}{2} \sum_N \sum_k \left(O_{k,n} - T_{k,n} \right)^2 + \frac{1}{2} \lambda \sum w^2$$

$$\frac{\partial E}{\partial w'} = \frac{\partial O_{k,n}}{\partial w'} \sum_N \sum_k \left(O_{k,n} - T_{k,n} \right) + \lambda w' \sum w$$

Linear outputs

Once we have a hidden layer, a nonlinear transfer function at the output is not necessary, nor desirable for modelling functions with an infinite range.

$$O_k = f(x_k) \quad \text{where} \quad x_k = \sum_{j=1}^{J} w_{j,k} H_j$$

$$f(x_k) = x_k \quad \text{rather than} \quad \frac{1}{1+e^{-\nu_x x_k}}$$

Input-to-hidden mapping is unchanged

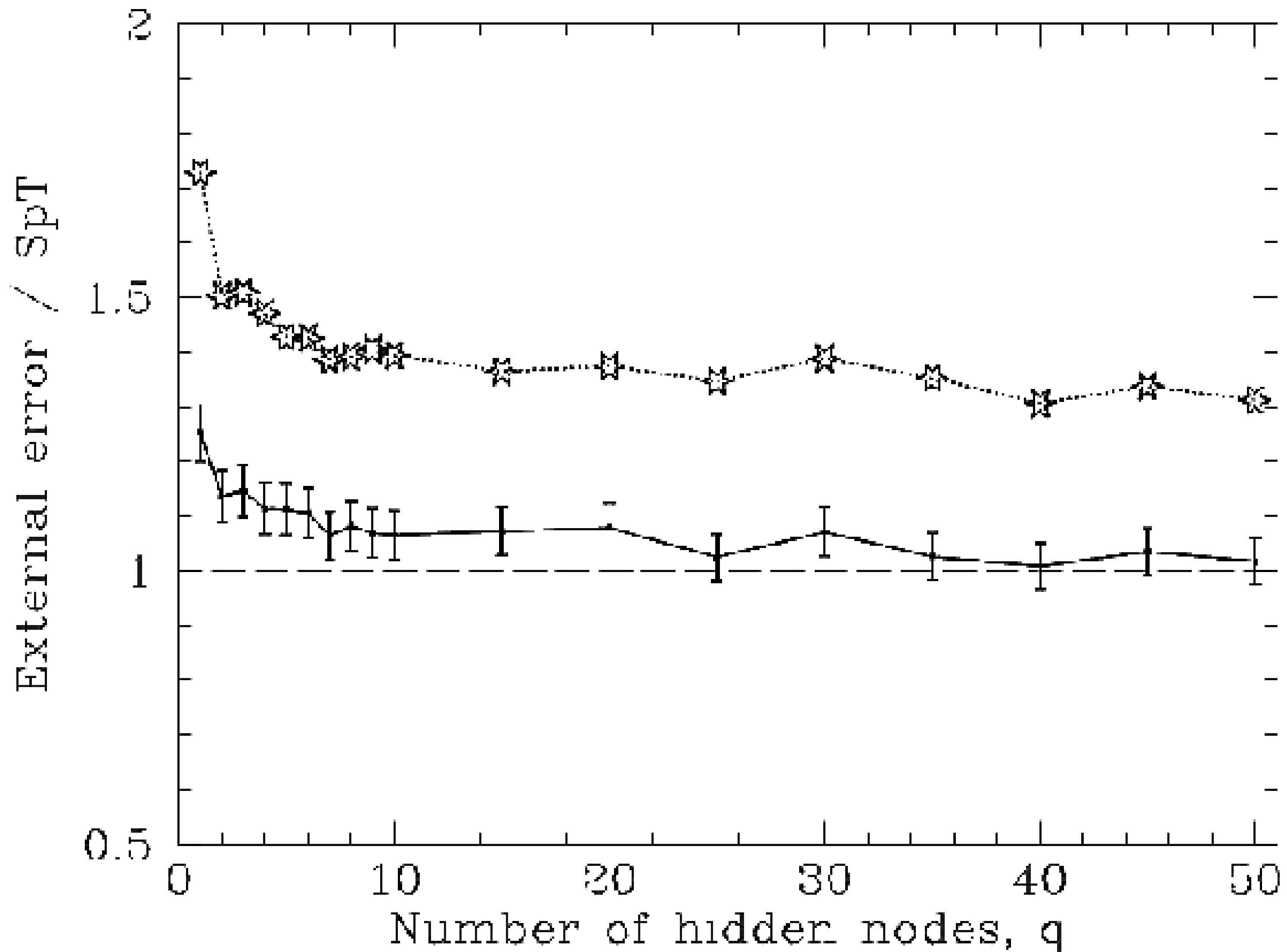
$$H_j = g(y_j) \quad \text{where} \quad y_j = \sum_i w_{i,j} I_i$$

$$g(y_j) = \frac{1}{1+e^{-\nu_y y_j}} \quad \text{sigmoidal transfer function}$$

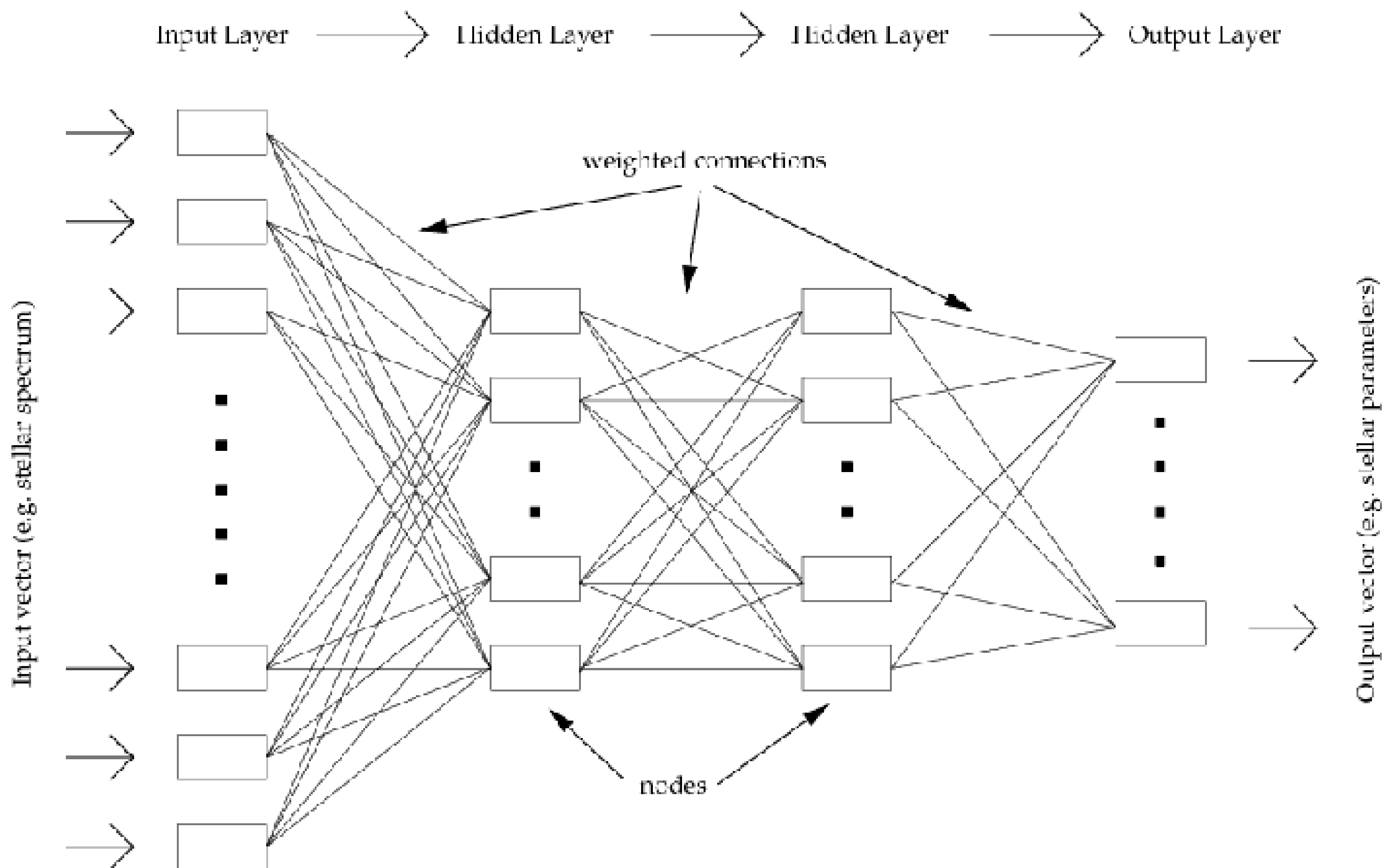
Putting it all together

$$O_k = \sum_{j=1}^{J} w_{j,k} \left(\frac{1}{1+\exp\left(-\nu_y \sum_i w_{i,j} I_i\right)} \right)$$

Impact of number of hidden nodes (~ complexity)



Two hidden layers



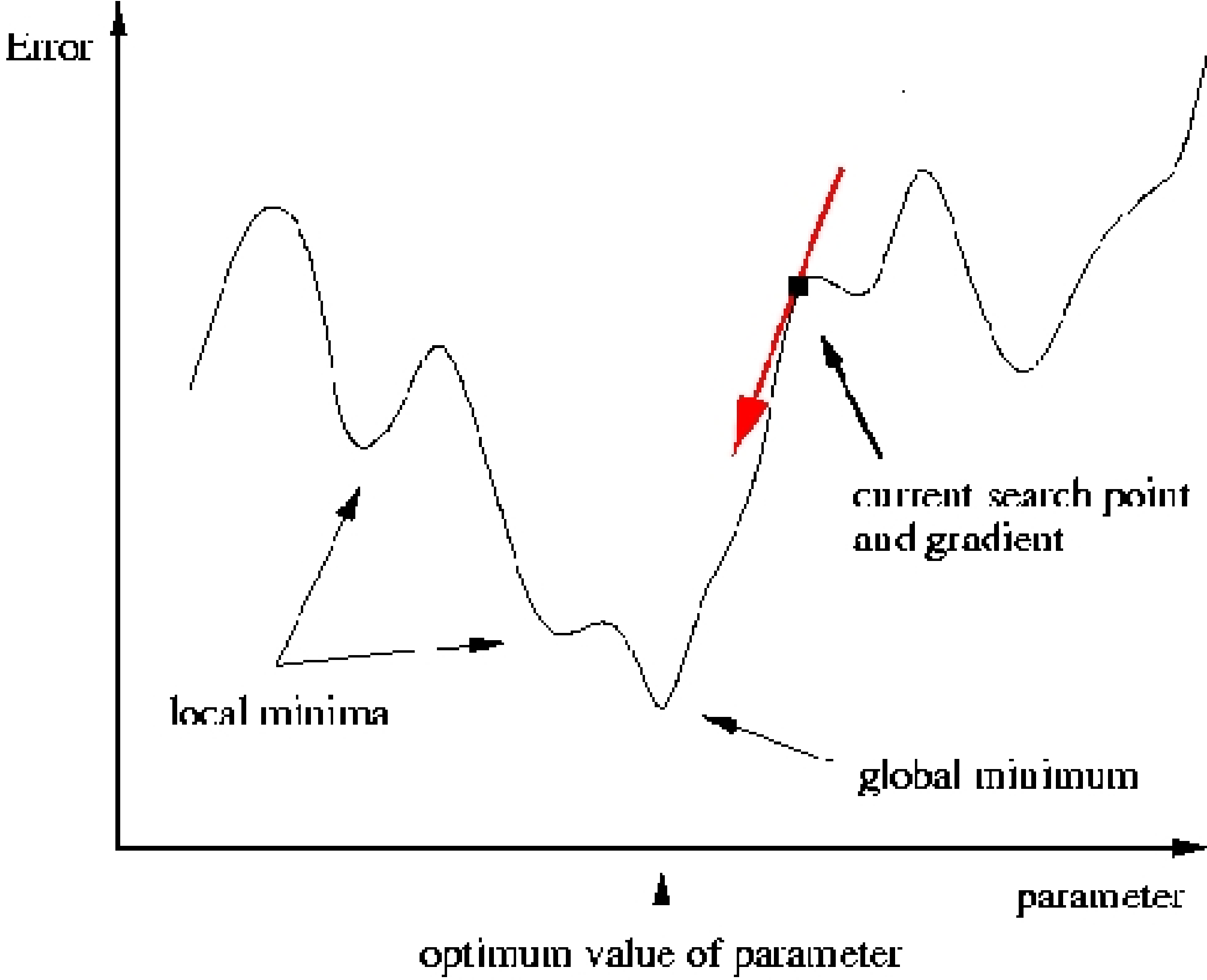
Network flexibility/complexity

- Flexibility is controlled by number of hidden nodes and number of hidden layers
  - in theory one hidden layer is sufficient: with sufficiently large J, network can approximate any continuous function to arbitrary accuracy (caveat generalization ability)
  - in practice using two means fewer weights
- Need sufficient training vectors to overdetermine solution
  - NK error functions
  - (I+1)J + (J+1)K weights
  - but in practice weights are not independent

Optimization algorithms

- with gradient information
  - gradient descent
  - add second derivative (Hessian): Newton, quasi-Newton, Levenberg-Marquardt
  - conjugate gradients (Hessian not explicitly calculated)
- pure gradient methods get stuck in local minima
  - random restart
  - committee/ensemble of models
  - momentum terms (non-gradient info.)
- without gradient information
  - simulated annealing
  - genetic algorithms

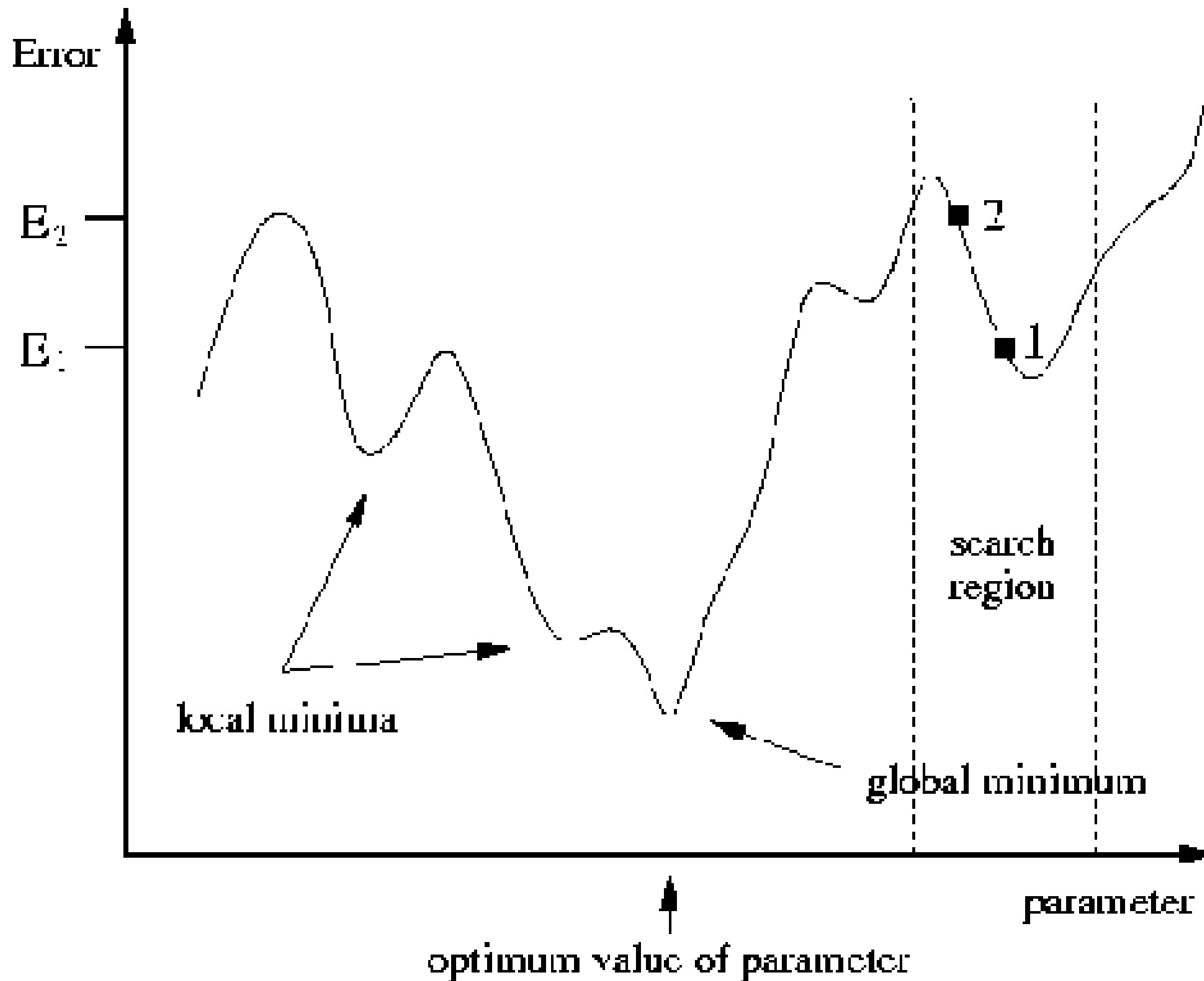
Local minima in optimization



Simulated Annealing

Probability of moving Δw from state 1 to state 2 is

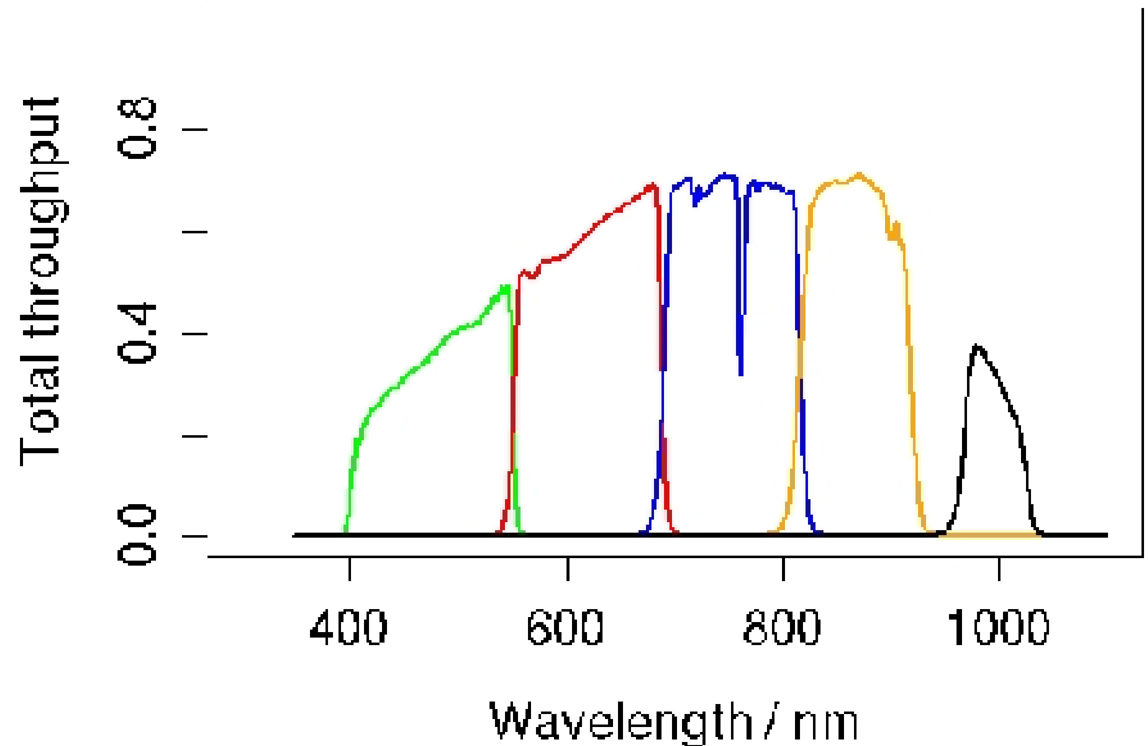
$$P(\Delta w \mid [E_2 - E_1]) = \begin{cases} 1 & \text{if} \quad E_2 < E_1 \\ \exp\left(\frac{E_1 - E_2}{kT}\right) & \text{otherwise} \end{cases}$$



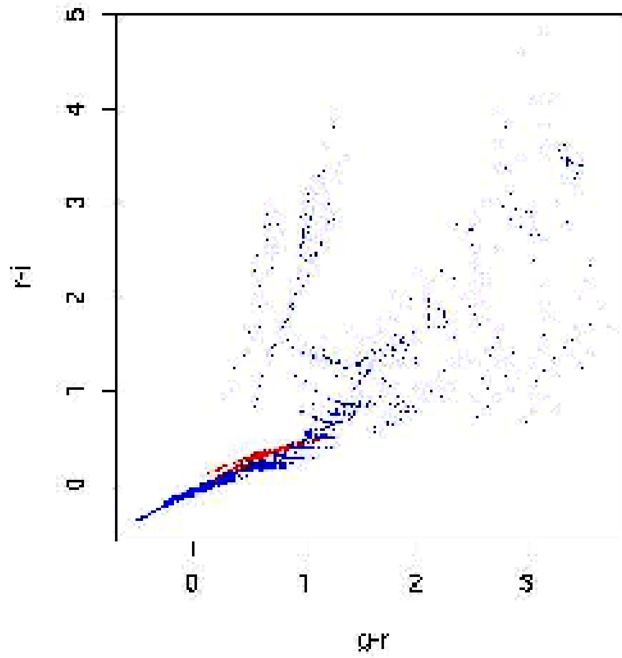
- candidate states chosen at random
- annealing temperature, T, reduced with time, thus forcing convergence
- analogy to statistical mechanics (Boltzmann formula)

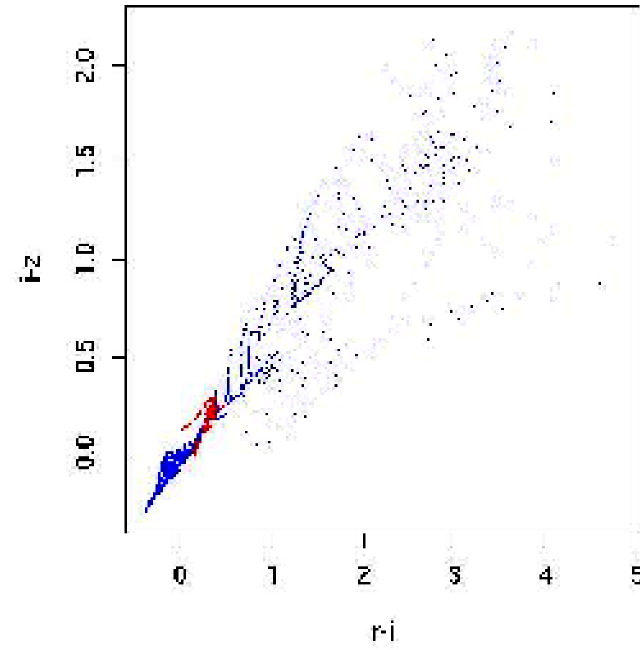
Neural network example: Stellar T_{eff} from PS1 photometry

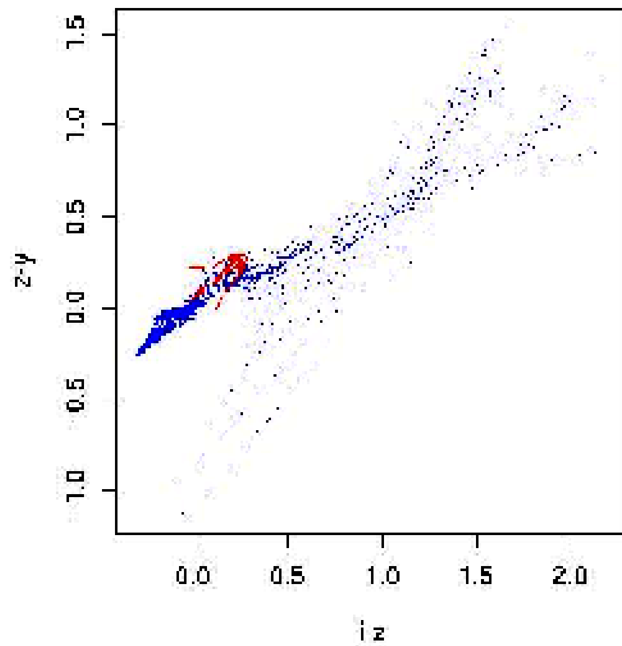
- Predict T_{eff} as a function of four optical colours
- Simulations of 8286 stellar spectra (noise-free)
- nnet{nnet} in R (see R scripts)
- 2-fold cross-validation



PS1 colours

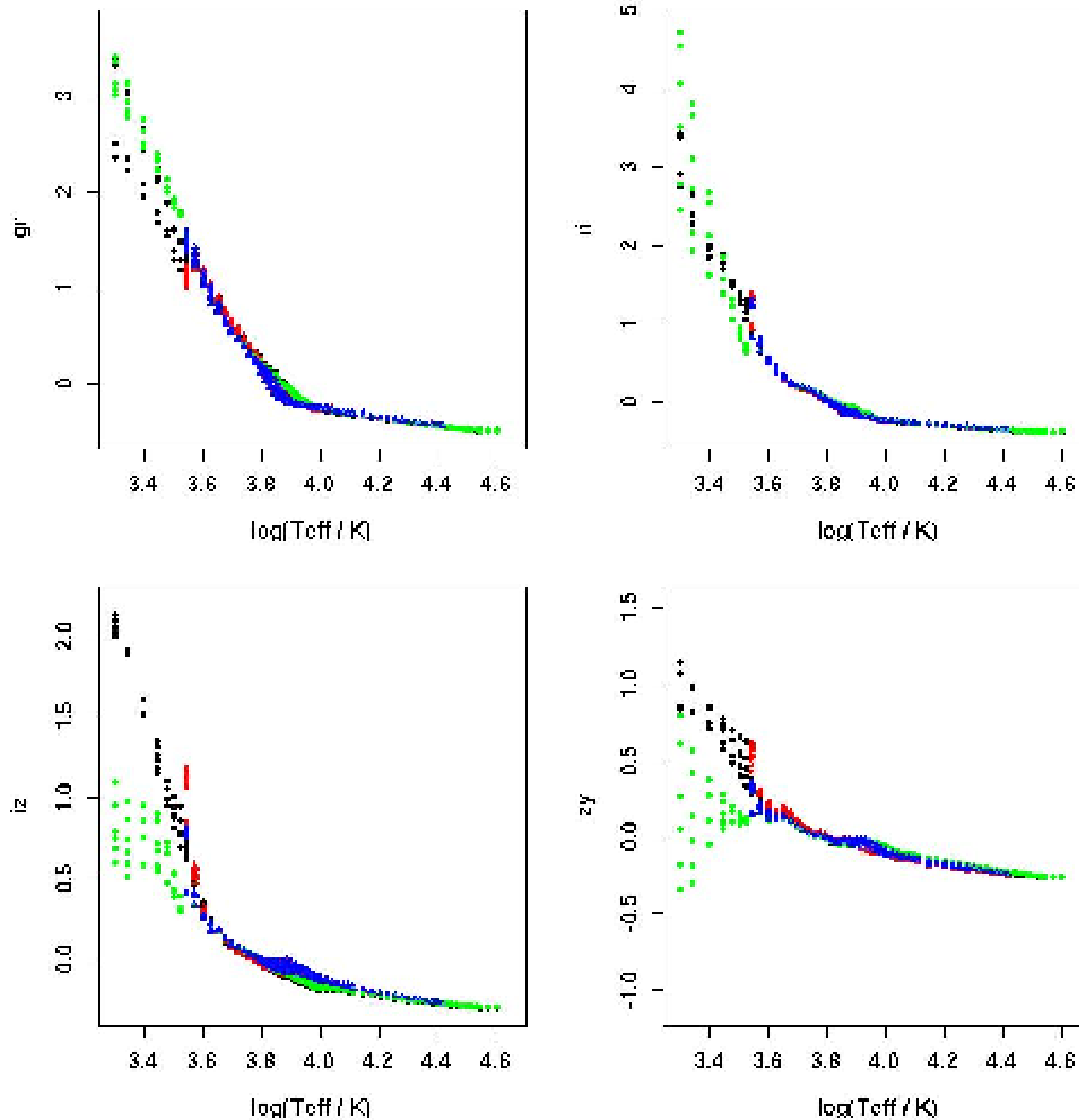




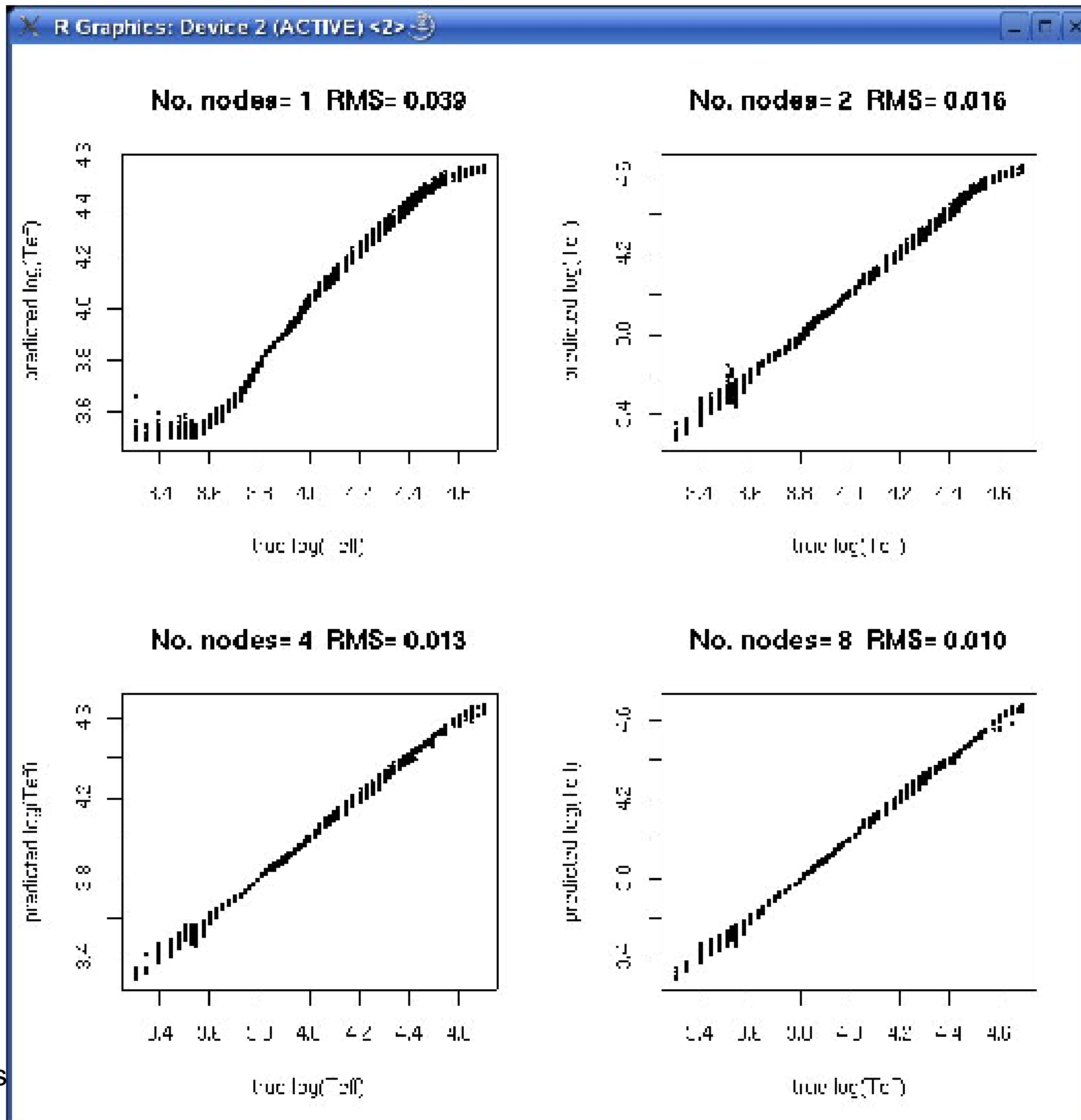


blue: stellar library
red: galaxy library

PS1 colour dependency on T_{eff} for a *limited* subset



Neural network results for T_{eff} prediction



Neural network modelling tips

- preprocess input and target data
  - reduce dynamic range (e.g. via logs)
  - standardize data and randomize initial weights on same, small scale, e.g. -1 to +1
  - ensure output function is appropriate to target range!
- try different numbers of hidden nodes
  - 5,10,20,50 hidden nodes; 1, 2 hidden layers
- training (parameter learning)
  - understand stopping (“convergence”) criterion
  - need to optimize weight decay parameter (e.g. via CV)
  - training on noise and early stopping are crude forms of regularization
- solution found depends on starting point
  - use a committee of 5-10 networks
  - global minimum rarely found in practice

Neural networks

- Nonlinear
  - sigmoidal transfer (basis) functions
  - nonlinearity introduced by interactions in the 'hidden' layer(s)
  - fit (“train”) numerically by minimizing a cost function
- Function modelling
  - flexibility determined by number of hidden nodes and layers
  - complexity control via regularization (e.g. “weight decay”)
- High dimensionality
  - the curse addressed via structured regression (transfer functions): no. parameters (weights) increases linearly with dimension

$$O_k = \sum_{j=1}^{J} w_{j,k} H_j$$

$$H_j = g(y_j) \quad \text{where} \quad y_j = \sum_i w_{i,j} I_i$$

$$g(y_j) = \frac{1}{1 + e^{-\nu_y y_j}} \quad \text{sigmoidal transfer function}$$