UiO : Matematisk institutt

Det matematisk-naturvitenskapelige fakultet

Neural Networks, Chapter 11 in ESL II

STK-IN4300
Statistical Learning Methods in Data Science

Odd Kolbjørnsen, oddkol@math.uio.no

Learning today Neural nets

- Projection pursuit
  - What is it?
  - How to solve it: Stagewise
- Neural nets
  - What is it?
  - Graphical display
  - Connection to Projection pursuit
  - How to solve it: Backprojection
  - Stochastic Gradient decent
  - Deep and wide
- Example

Neural network

- Used for prediction
- Universal approximation
  - with enough data and the correct algorithm you will get it right eventually…
- Used for both «regression type» and «classification» type problems
- Many versions and forms, currently deep learning is a hot topic
- Perform highly advanced analysis
- Can create utterly complex models which are hard to decipher and hard to use for knowledge transfer.
- The network provide good prediction, but is it for the right reasons?
- Often portrayed as fully automatic, but tailoring might help

In neural nets training is based on minimization of a loss function over the training set

General form
$$L\left(Y, \hat{f}(X)\right) = \sum_{i=1}^{N} L(y_i, \hat{f}(x_i))$$

Neural nets are defined by a specific form of the model $f(X)$

- Target might be multi dimensional

$$y_i = [y_{i1}, \dots, y_{iK}]^T$$

- Continuous response («regression type»)

Squared error (common)
$$L\left(Y, \hat{f}(X)\right) = \sum_{i=1}^{N}\sum_{k=1}^{K}\left(y_{ik} - \hat{f}_k(x_i)\right)^2$$

- Discrete (K –classes)

Squared error
$$L\left(Y, \hat{f}(X)\right) = \sum_{i=1}^{N}\sum_{k=1}^{K}\left(y_{ik} - \hat{f}_k(x_i)\right)^2$$

Cross-entropy or deviance
$$L\left(Y, \hat{f}(X)\right) = \sum_{i=1}^{N}\sum_{k=1}^{K} -\log \hat{f}_k(x_i) \cdot y_{ik}$$

$$\hat{f}_k(x_i) \approx \mathrm{Prob}(y_{ik} = 1)$$
$$y_{ik} = \begin{cases} 0 & \text{if } y_i \neq k \\ 1 & \text{if } y_i = k \end{cases}$$

Projection pursuit Regression

Friedman and Tukey (1974)
Friedman and Stuetzle (1981)

Unknown functions ($\mathbb{R} \to \mathbb{R}$)

$$f(X) = \sum_{m=1}^{M} g_m(\underbrace{w_m^T X})$$

Derived feature (number m),
$V_m = w_m^T X$, (size 1×1)

Unknown Weight (size $p \times 1$) Unit vector

Features = Explanatory Variables (size $p \times 1$)

Ridge functions are constant along directions orthogonal to the directional unit vector w

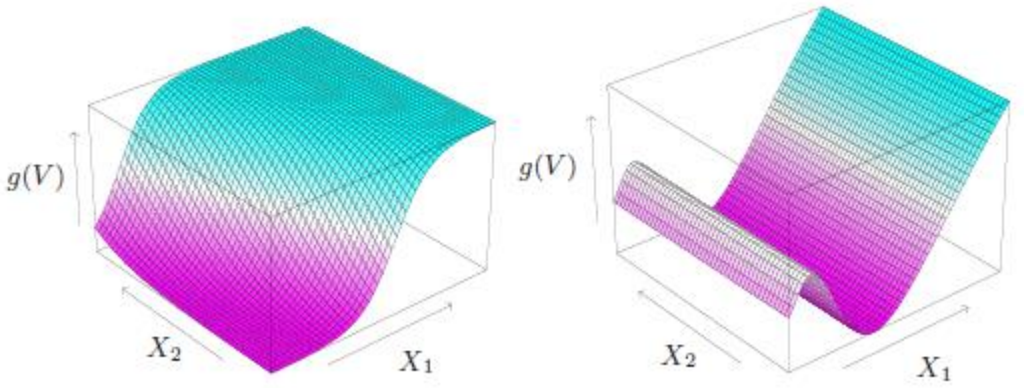


FIGURE 11.1. *Perspective plots of two ridge functions.*
(Left:) $g(V) = 1/[1 + \exp(-5(V - 0.5))]$, *where* $V = (X_1 + X_2)/\sqrt{2}$.
(Right:) $g(V) = (V + 0.1)\sin(1/(V/3 + 0.1))$, *where* $V = X_1$.

Fitting Projection pursuit: M=1

- M = 1 model, known as the single index model in econometrics:
  - $f(X) = g(w^T X) = g(V), \quad V = w^T X$

Iterate

- If w is known fitting $\hat{g}(v)$ is just a 1D smoothing problem
  - Smoothing spline, Local linear (or polynomial) regression, Kernel smoothing, K-nearest…
- If $g()$ is known fitting $\hat{w}$ is obtained by quasi-Newton search
  - $g(w^T x_i) \approx g\left(w_{\text{old}}^T x_i\right) + g'(w_{\text{old}}^T x_i)\left(w - w_{\text{old}}^T\right)^T x_i$
  - Minimize the objective function (with approximation inserted)

$$\sum_{i=1}^{N}\left(y_i - g(w^T x_i)\right)^2 \approx \sum_{i=1}^{N}\left(y_i - g\left(w_{\text{old}}^T x_i\right) - g'(w_{\text{old}}^T x_i)\left(w - w_{\text{old}}^T\right)^T x_i\right)^2$$

$$= \sum_{i=1}^{N} g'\left(w_{\text{old}}^T x_i\right)^2 \left(\frac{y_i - g\left(w_{\text{old}}^T x_i\right)}{g'\left(w_{\text{old}}^T x_i\right)} + w_{\text{old}}^T x_i - w^T x_i\right)^2$$

Only unknown

Solve for w using weighted regression:
weight $= g'\left(w_{\text{old}}^T x_i\right)^2$

Fitting Projection pursuit, $M > 1$

$$\boxed{f(X) = \sum_{m=1}^{M} g_m(w_m^T X)}$$

- Stage-wise (greedy)
  - Set $y_{i,1} = y_i$
  - For $m = 1, \ldots, M$
    - Assume there is just one function to match (as previous page)
    - Minimize Loss with respect to $y_{i,m}$ to obtain $g_m()$ and w_m

$$[\hat{g}_m(\cdot), \hat{w}_m] = \underset{g_m(\cdot), w_m}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_{i,m} - g_m(w_m^T x_i)\right)^2$$

    - Store $\hat{g}_m(\cdot)$ and $\hat{w}_m$
    - Subtract estimate from data $y_{i,m+1} = y_{i,m} - \hat{g}_m(\hat{w}_m^T x_i)$
  - Final prediction:

$$\hat{f}(X) = \sum_{m=1}^{M} \hat{g}_m(\hat{w}_m^T X)$$

Implementation details

1. Need a smoothing method with efficient evaluation of $g(v)$ and $g'(v)$
   - Local regression or smoothing splines

2. $g_m(v)$ from previous steps can be readjusted using a backfitting procedure (Chapter 9), but it is unclear if this improves the performance
   1. Set $r_i = y_i - \hat{f}(x_i) + \hat{g}_m(\hat{w}_m x_i)$
   2. Re-estimate $g_m(\cdot)$ from r_i. (and center the results)
   3. Do this repeatedly for $m = 1, \dots M, 1 \dots M, \dots$

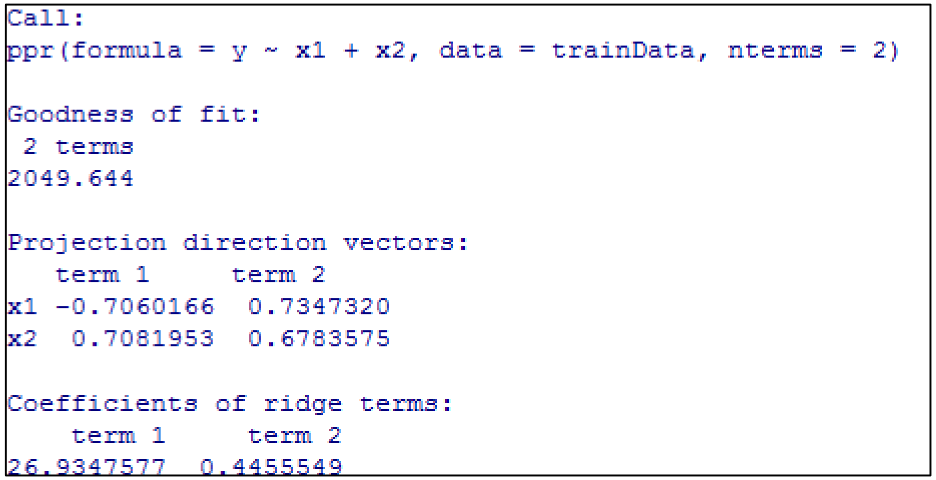
3. It is not common to readjust $\hat{w}_m$, as this is computationally demanding

4. Stopping criterion for number of terms to include.
   1. When the model does not improve appreciably
   2. Use cross validation to determine M

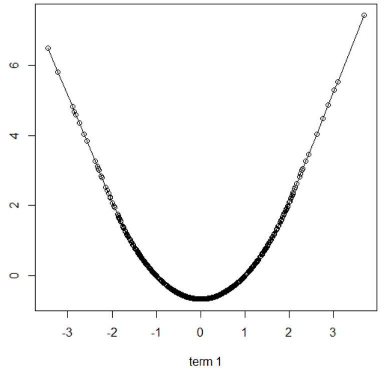
Example

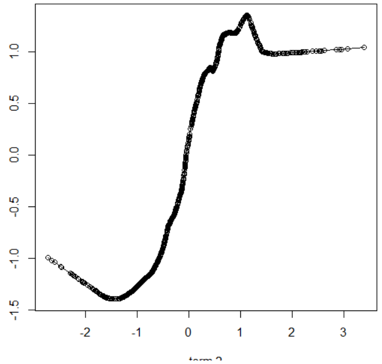
$$Y = \sigma(a_1^T X) + (a_2^T X)^2 + 0.30 \cdot Z, \qquad a_1 = (3,3),\ a_2 = (3,-3);$$

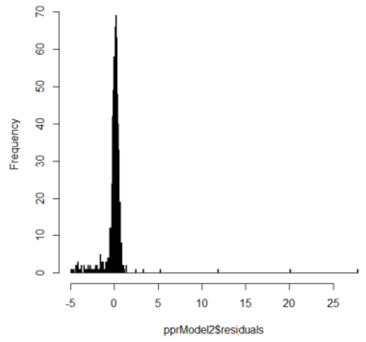
- Train data: 1000
- Two terms:



```
Call:
ppr(formula = y ~ x1 + x2, data = trainData, nterms = 2)

Goodness of fit:
 2 terms
2049.644

Projection direction vectors:
   term 1     term 2
x1 -0.7060166  0.7347320
x2  0.7081953  0.6783575

Coefficients of ridge terms:
    term 1     term 2
26.9347577  0.4455549
```



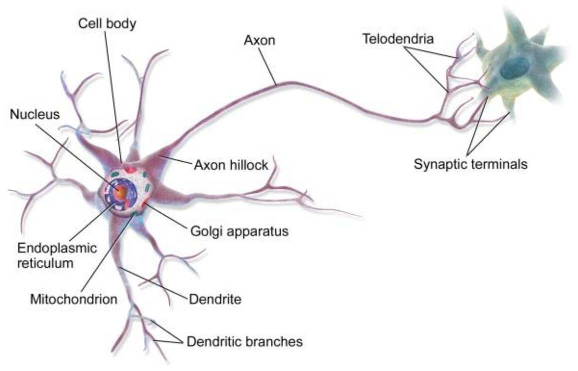




Neural networks

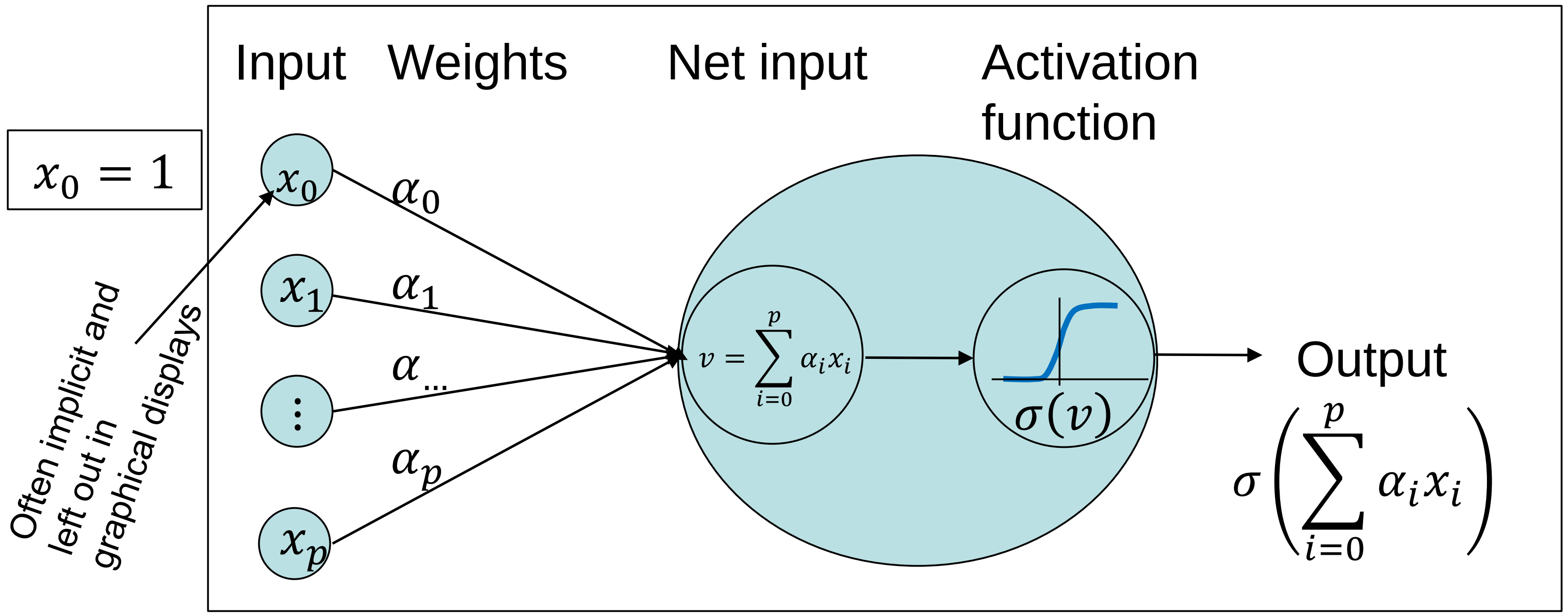
- Deep neural nets are currently «hot-topic»
- Deep means many hidden layers
- Multilayer feed forward
  - Characteristics
    - Network is arranged in layers,
      - first layer taking input
      - last layer outputs
      - Intermediate layers are hidden layers (no connection to the world outside)
    - A node in one layer is connected to every node in the next layer (but the weight can be zero)
    - There are no connections among nodes in the same layer
- Other types:
  - Self organizing map (SOM), output is not defined (unsupervised)
  - Recurrent neural network (RNN), many forms
  - Hopfield networks (RNN with symmetric connections)
  - Boltzmann machine networks (Markov random fields)

Neural network



- Simplified model of a nerve system

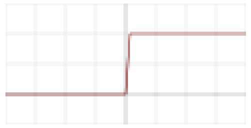
Perceptron:



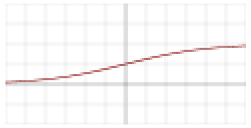
Activation functions

$\sigma(v)$

- Initially: The binary step function used

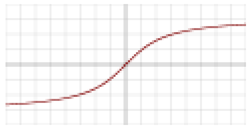


- Next: Sigmoid = Logistic = Soft step



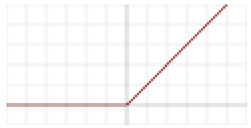
Smooth

- Now: there is a «rag bag» of alternatives some more suited than others for specific tasks
  - ArcTan



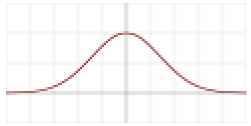
Smooth

  - Rectified linear ReLu



Continious

  - Gaussian (NB not monotone gives different behavior)



Smooth

Illustrations from:
https://en.wikipedia.org/wiki/Activation_function

Single layer feed-forward Neural nets

Activation function ($\mathbb{R} \to \mathbb{R}$)

$$f(X) = \sum_{m=1}^{M} \beta_m \sigma(\underbrace{\alpha_m^T X + \alpha_0})$$

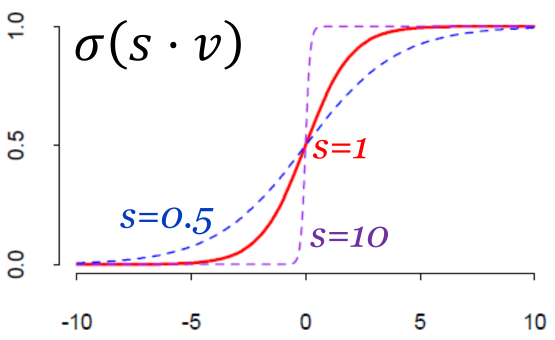
Derived feature (number m),
$Z_m = \alpha_m^T X + \alpha_0$, (size 1 × 1)

Unknown Weight (size (p × 1) **Not** unit vector

Feature = Explanatory variables (size p × 1)

«Bias» or «Shift»

Sigmoid



scale unit vector

$$\sigma(\alpha^T X + \alpha_0) = \sigma(s_m \cdot w_m^T X + \alpha_0) = \sigma(s_m \cdot V_m + \alpha_0)$$

$$\boxed{w_m = \frac{\alpha_m}{s_m}, \qquad s_m = \|\alpha_m\|}$$

«PP – Feature»

Graphical display of single hidden layer feed forward neural network

$$f_k(X) = \sum_{m=1}^{M} \beta_{k,m}\sigma(\alpha_m^T X + \alpha_0)$$

Note!
With respect to model definition

Feed forward means:
- Connections in the graph are directional
- The direction goes from input to output

11.3 Neural Networks 393

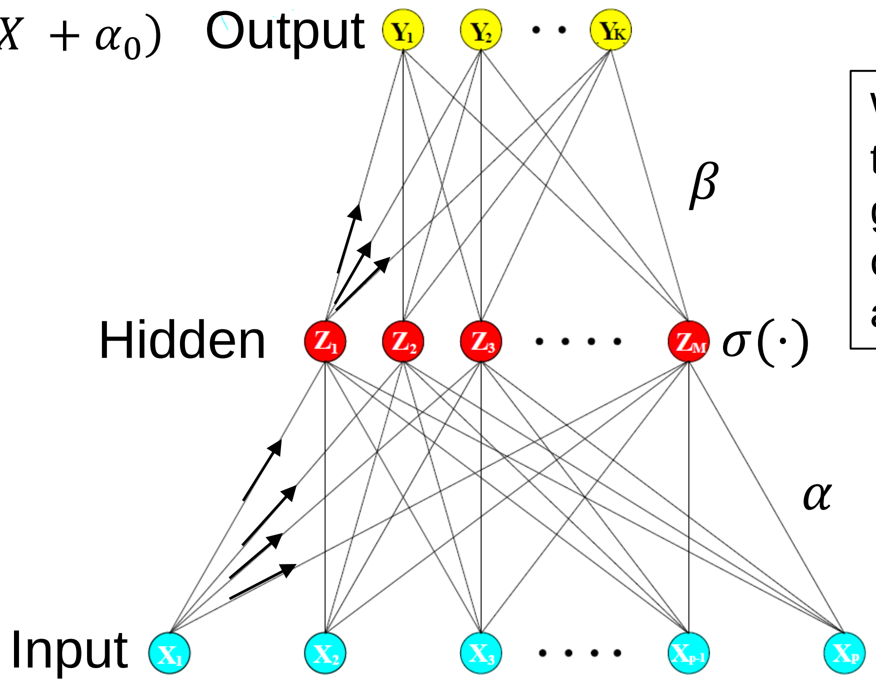


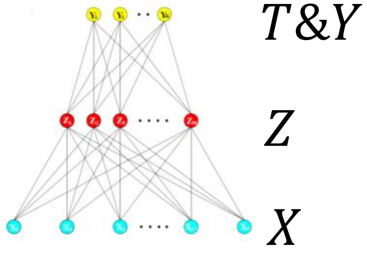
FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

We will however traverse the graph in the opposite direction as well ….

Output layer is often «different»

Hidden layer: $Z_m = \sigma\left(\alpha_{0,m} + \alpha_m^T X\right), \quad m = 1, \dots M$

Output layer: $T_k = \beta_{0,k}, +\beta_k^T Z, \quad k = 1 \dots K$



Some alternatives for $f_k()$:

Transform	$\sigma(T_k)$	Same as «hidden» layers
Identity	T_k	Common in regression setting
Joint transform	$g_k\ (T)$	Common for classification, e.g. softmax

Identity

$$f_k(X) = T_k$$
$$= \beta_{0,k} + \sum_{m=1}^{M} \beta_{k,m}\sigma\left(\alpha_m^T X\ + \alpha_{0,m}\right)$$

Softmax

$$f_k(X) = \frac{\exp(T_k)}{\sum_{j=1}^{K} \exp(T_j)}$$
$$= \frac{\exp\left(\beta_{0,k}, +\beta_k^T Z\right)}{\sum_{j=1}^{K} \exp(\beta_{0,j}, +\beta_j^T Z)}$$

Comparision Projection pursuit (PP) and Neural nets (NN)

$$\boxed{f(X) = \sum_{m=1}^{M_{PP}} g_m(w_m^T X)}$$

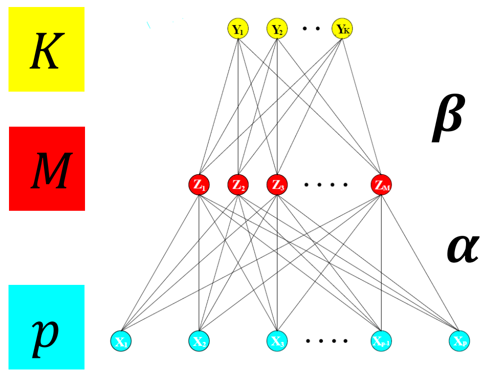
$$\boxed{f(X) = \sum_{m=1}^{M_{NN}} \beta_m \sigma(\alpha_m^T X + \alpha_0)}$$

$$g_m(w_m^T X) \quad \text{vs} \quad \beta_m \sigma(s_m \cdot w_m^T X + \alpha_0)$$

$$\boxed{s_m = ||\alpha||}$$

- The flexibility of g_m is much larger than what is obtained with s_m and α_0 which are the additional parameters of neural nets
- There are usually less terms in PP than NN, i.e. $M_{PP} \ll M_{NN}$
- Both methods are powerful for regression and classification
- Effective in problems with high signal to noise ratio
- Suited for prediction without interpretation
- Identifiability of weights an open question and creates problems in interpretations
- The fitting procedures are different

Fitting neural networks



θ: Statistical slang for all parameters

Here:

$\{\alpha_{0,m}, \alpha_m\}$, # parameters: $(p+1)M$

$\{\beta_{0,m}, \beta_m\}$, # parameters: $(M+1)K$

The "standard" approach:

- Minimize the loss
- Use steepest decent to solve this minimization problem

- The key to success is the efficient way of computing the gradient

$$R(\theta) = L\left(Y, \hat{f}(X)\right)$$

Quadratic loss
K output varaibles

$$= \sum_{i=1}^{N}\sum_{k=1}^{K}\left(y_{ik} - \hat{f}_k(x_i)\right)^2$$

$$= \sum_{i=1}^{N} R_i(\theta)$$

Contribution of
the i'th data record

$$R_i(\theta) = \sum_{k=1}^{K}\left(y_{ik} - \hat{f}_k(x_i)\right)^2$$

Steepest decent

- Minimize $R(\theta)$ wrt θ,
  - Initialize: $\theta^{(0)}$
  - Iterate:

$$\theta_j^{(r+1)} = \theta_j^{(r)} - \underbrace{\gamma_r}_{\text{Learning rate}} \left.\frac{\partial R(\theta)}{\partial \theta_j}\right|_{\theta=\theta^{(r)}}$$

$$\frac{\partial R(\theta)}{\partial \theta_j} = \sum_{i=1}^{N} \frac{\partial R_i(\theta)}{\partial \theta_j}$$

we compute term per data record $\frac{\partial R_i(\theta)}{\partial \theta_j}$ (easily aggregated from parallel computation)

Squared error loss

$$f_k(X) = g_k(T)$$

Output layer:

$$\frac{\partial R_i(\theta)}{\partial \beta_{k,m}} = \underbrace{-2\left(y_{i,k} - f_k(x_i)\right) g_k'(\beta_k^T z_i)}_{\delta_{k,i}} z_{m,i}$$

$$= \delta_{k,i} \cdot z_{m,i}$$

Hidden layer:

$$\frac{\partial R_i(\theta)}{\partial \alpha_{m,l}} = \underbrace{-\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i)\beta_{km}\, \sigma'(\alpha_m^T x_i)}_{s_{m,i}} x_{i,l}$$

$$= s_{m,i} \cdot x_{i,l}$$

Back propagation equation

$$s_{m,i} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{k,i}$$

Back propagation (delta rule)

- At top level. compute:

$$\delta_{k,i} = -2\left(y_{i,k} - f_k(x_i)\right) g_k'\left(\beta_k^T z_i\right), \qquad \forall(i,k)$$

- At hidden level, compute:

$$s_{m,i} = \sigma'\left(z_{m,i}\right) \sum_{k=1}^{K} \beta_{k,m} \delta_{k,i}, \qquad \forall(i,m)$$

- Evaluate:

$$\frac{\partial R_i(\theta)}{\partial \beta_{k,m}} = \delta_{k,i} z_{m,i} \;\;\&\;\; \frac{\partial R_i(\theta)}{\partial \alpha_{m,l}} = s_{m,i} x_{i,l}$$

- Update : γ_r is fixed

$$\beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i=1}^{N} \left.\frac{\partial R_i}{\partial \beta_{k,m}}\right|_{\theta=\theta^{(r)}}$$

$$\alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i=1}^{N} \left.\frac{\partial R_i}{\partial \alpha_{m,l}}\right|_{\theta=\theta^{(r)}}$$

Stochastic gradient decent

$$\beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{k,m}}\bigg|_{\theta=\theta^{(r)}} \quad \alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m,l}}\bigg|_{\theta=\theta^{(r)}}$$

- Equations above updates with all data at the same time
- The form invites to update estimate using fractions of data
  - Perform a random partition of training data in to batches: $\{B_j\}_{j=1}^{\#\text{Batches}}$
  - For all batches cycle over the data in this batch to update data

$$\beta_{k,m}^{(r+1)} = \beta_{k,m}^{(r)} - \gamma_r \sum_{i\in B_j} \frac{\partial R_i}{\partial \beta_{k,m}}\bigg|_{\theta=\theta^{(r)}} \quad \alpha_{m,l}^{(r+1)} = \alpha_{m,l}^{(r)} - \gamma_r \sum_{i\in B_j} \frac{\partial R_i}{\partial \alpha_{m,l}}\bigg|_{\theta=\theta^{(r)}}$$

  - Repeat

- On iteration is one update of the parameter (using one batch)
- One Epoch is one scan through all data (using all batches in the partition)

Online learning (Extreme case Batch size =1)

- Learning based on one data point at the time

$$\beta_{k,m}^{(r)} = \beta_{k,m}^{(r-1)} - \gamma_r \left.\frac{\partial R_i}{\partial \beta_{k,m}}\right|_{\theta=\theta^{(r-1)}}$$
$$\alpha_{m,l}^{(r)} = \alpha_{m,l}^{(r-1)} - \gamma_r \left.\frac{\partial R_i}{\partial \alpha_{m,l}}\right|_{\theta=\theta^{(r-1)}}$$

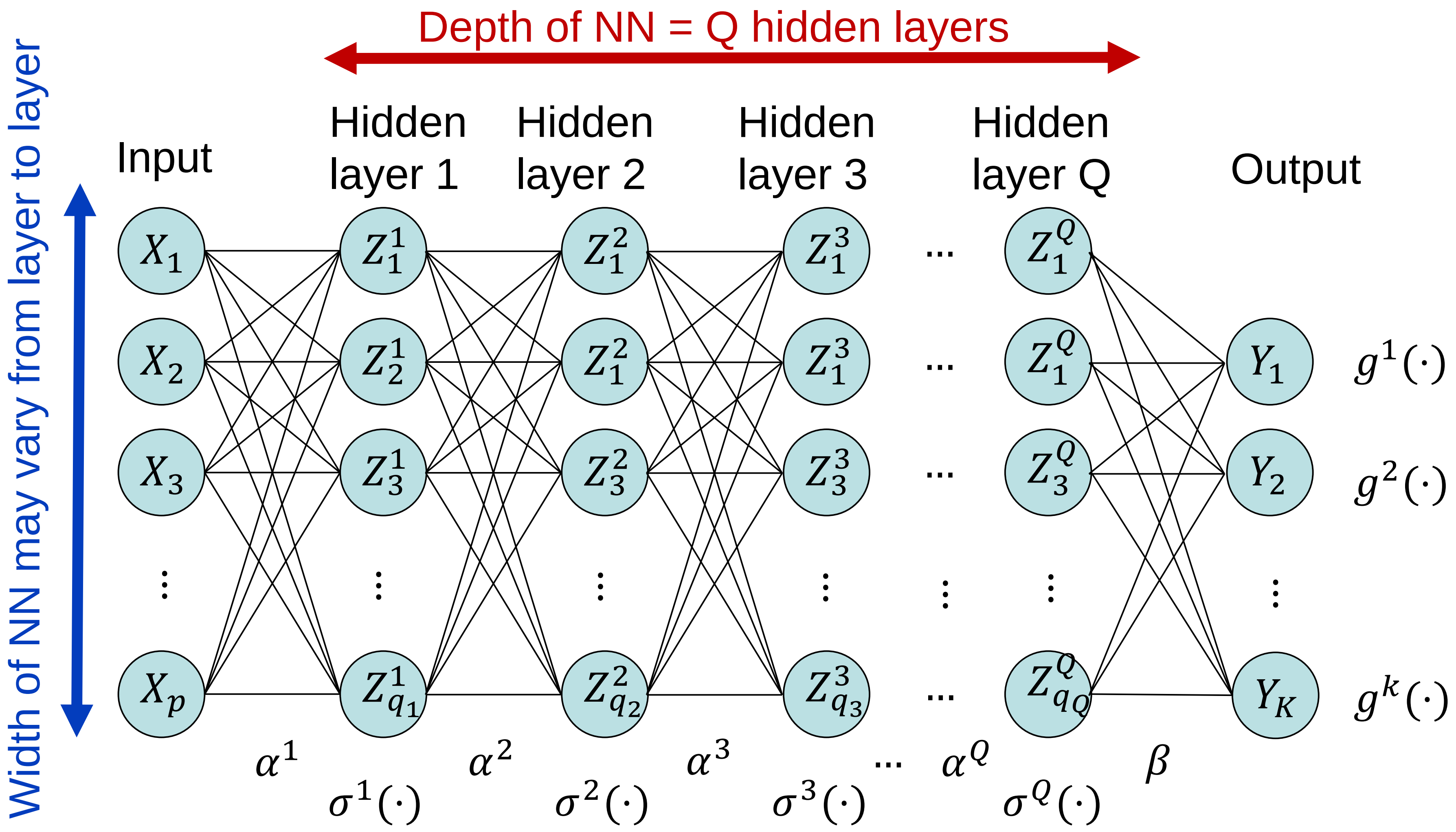
- You might re-iterate (for several epochs) when completed or if you have an abundance of data just take on new data as they come along (hence the name)

- For convergence: $\gamma_r \to 0$, as $\sum \gamma_r \to \infty$ and $\sum \gamma_r^2 < \infty$, e.g. $\gamma_r = \frac{1}{r}$

Other methods can be used

- Still use the Backpropagation to get the derivative
- Conjugate Gradient
  - Method for minimizing a quadratic form
  - Need «restart» for nonlinear problems
- Variable metric methods
  - E.g. Quasi newton methods

Graphical display of feed forward neural network



Nested definition (Do not try to write this in a closed form…)

$$f_k(X) = \beta_{0,k} + \sum_{m=1}^{q_Q} \beta_{m,k} Z^{[Q]}$$

Number of outputs
$k = 1, \dots, K$

$$Z_i^{[Q]} = \sigma^{Q-1}\left({\alpha_i^{[Q]}}^T \cdot Z^{[Q-1]} + \alpha_{0,i}^{[Q]} \right)$$

size: $(1 \times q_{Q-1})$

size: (1×1)

Width of layer Q
$i = 1, \dots, q_Q$

$$Z_i^{[Q-1]} = \sigma^{Q-2}\left({\alpha_i^{[Q-1]}}^T \cdot Z^{[Q-2]} + \alpha_{0,i}^{[Q-1]} \right)$$

$i = 1, \dots, q_{Q-1}$

$$\vdots$$

$$Z_i^{[1]} = \sigma^{1}\left({\alpha_i^{1}}^T \cdot X + \alpha_0^{1} \right)$$

size: $(1 \times p)$

size: (1×1)

Number of input

Width of layer 1
$i = 1, \dots, q_1$

Training Neural networks

- Back propagation can still be used in a recursive way. Traversing the graph backwards, generalizing the expression:

$$s_{m,i} = \sigma'(z_{m,i}) \sum_{k=1}^{K} \beta_{k,m} \delta_{k,i}\,, \qquad \forall(i,m)$$

- See e.g. Schmidhuber 2015, with references

Scaling of input & Starting values

- Standardize input variables to avoid numerical scaling issues
  - Mean 0
  - Standard deviation 1
- Close to zero (model almost linear)
- Do not choose zero (then it does not get started)
- Too large values generally gives bad results
- Common to use one (or several) random starting point
- With standardized input a common choice is to draw weights from a uniform distribution uniform in $[-0.7 \;\; 0.7]$

Overfitting

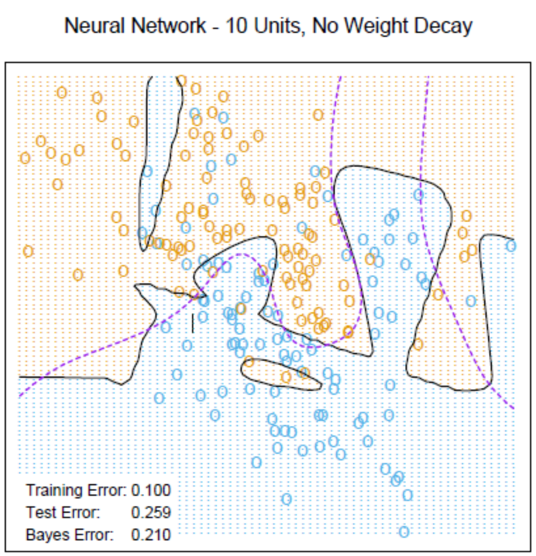
- Early stopping
  - Since starting is close to linear one will usually end up with something close to linear
  - Use a validation set to select when to stop
- Regularization by weight decay
  - Minimize: $R(\theta) + \lambda J(\theta) \longleftarrow$ Penalty term

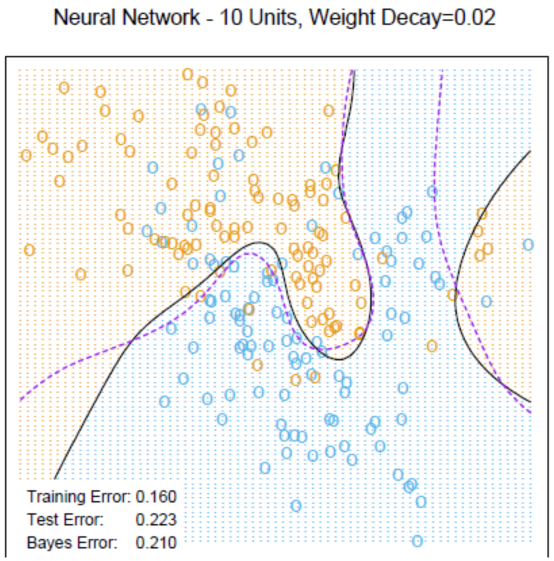
$$J(\theta) = \sum \alpha_{m,l}^2 + \sum \beta_{k,m}^2$$

  - Use cross validation to select λ

Effect of weight decay

Prediction

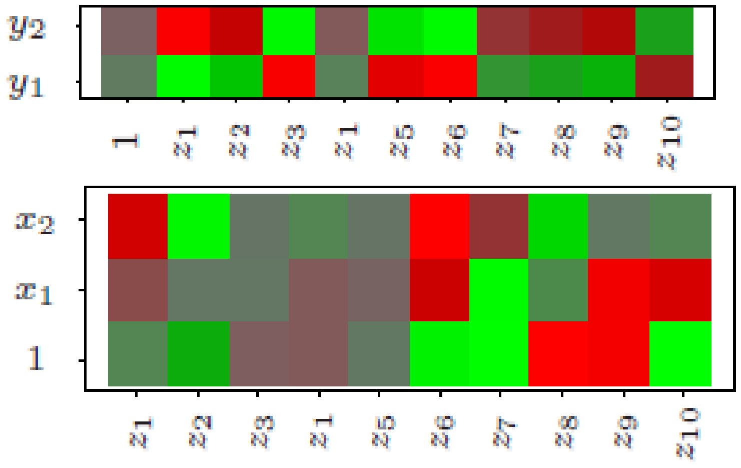


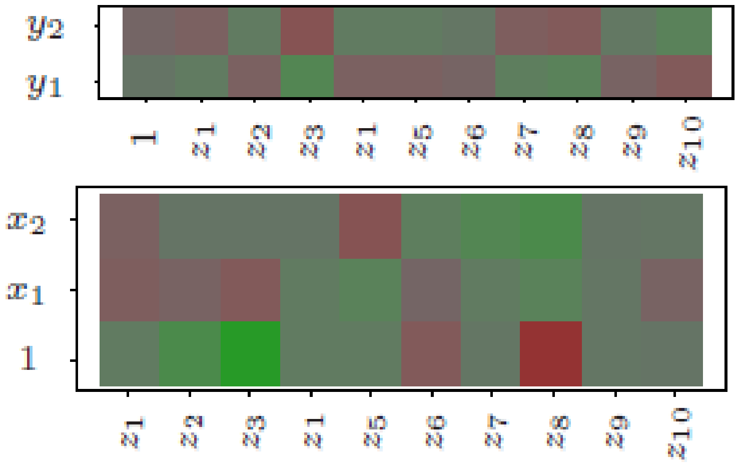


No weight decay

Weight decay

Weights





Multiple Minima

- $R(\theta)$ is not convex and have many local minima
- Try many starting configurations
  - choose solution with lowest penalize error
  - Use the average prediction of the collection of networks
    - NB do not average weights as these are not well ordered
- Use bagging, i.e. average predictions of networks trained from random perturbations of training data

Number of hidden Units and layers

- Number of units
  - From 5-100 units is common
  - Increase with number of input and training data
  - Better to have too many than to few
  - Start with a large number and use weight decay (regularization)
- Number of hidden layers
  - Guided by background knowledge
  - Models hierarchical features at different levels of resolution
  - Trial and error

What is Deep learning

- There is no universally agreed upon threshold of depth dividing shallow learning from deep learning, but most researchers in the field agree that deep learning has multiple nonlinear layers (CAP > 2), 3 layers and more.

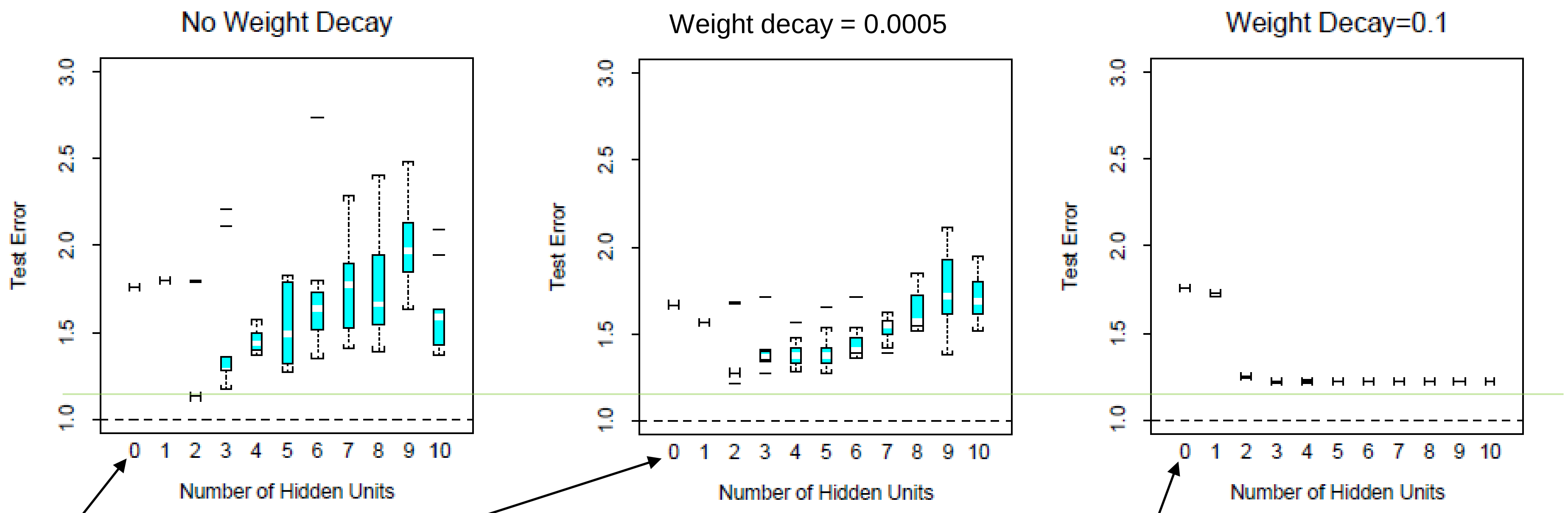
- «Deep learning» Hinton et al 2006 (3 layers)
- «Very Deep learning» Simonyan et al. 2014 (16+ layers)
- «Extremely Deep» He et al 2016 (50 -> 1000)
- Schmidhuber 2015 considers more than 10 layers to be very deep learning

Example simulated data

Sum of sigmoids: $Y = \sigma(a_1^T X) + \sigma(a_2^T X) + \varepsilon_1;$

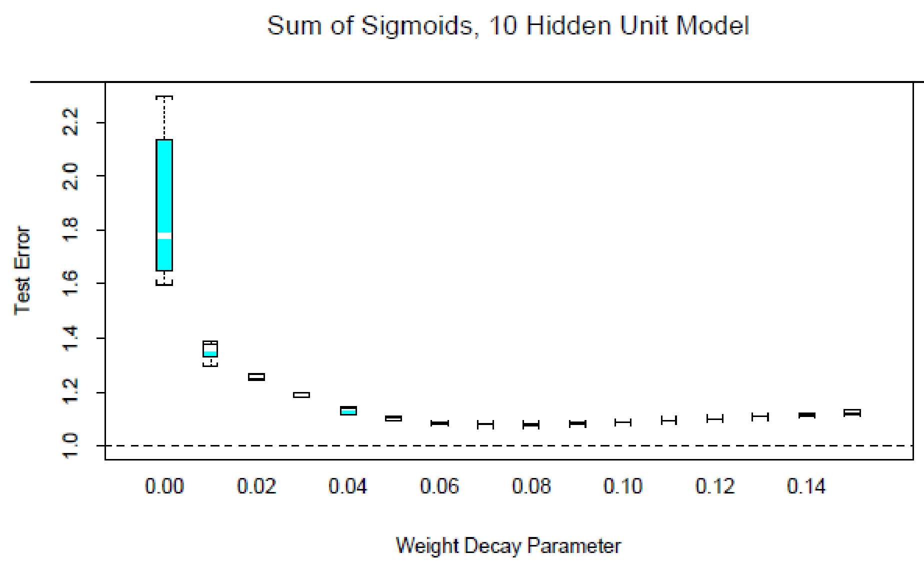
$a_1 = (3,3),\ a_2 = (3,-3);$

- $\frac{\mathrm{Var}(f(X))}{\mathrm{Var}(\varepsilon_1)} \approx 4$
- Training data size: 100 samples
- Test data size : 10 000 samples

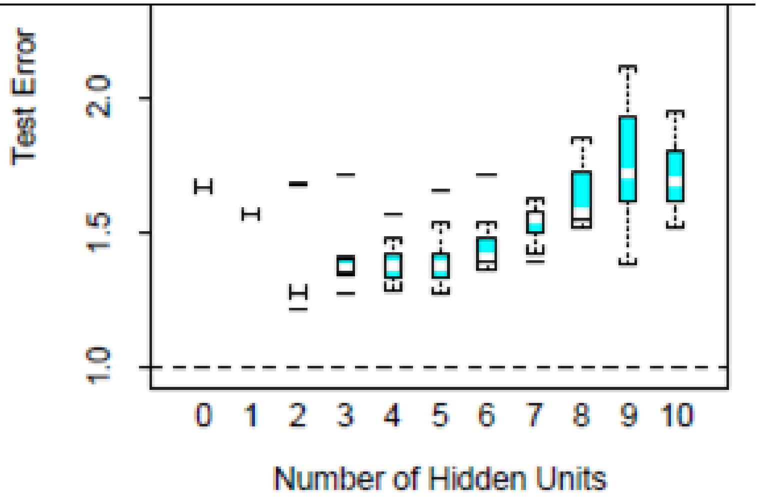


Linear regression

Weight decay at 10 hidden units



Weight decay = 0.0005



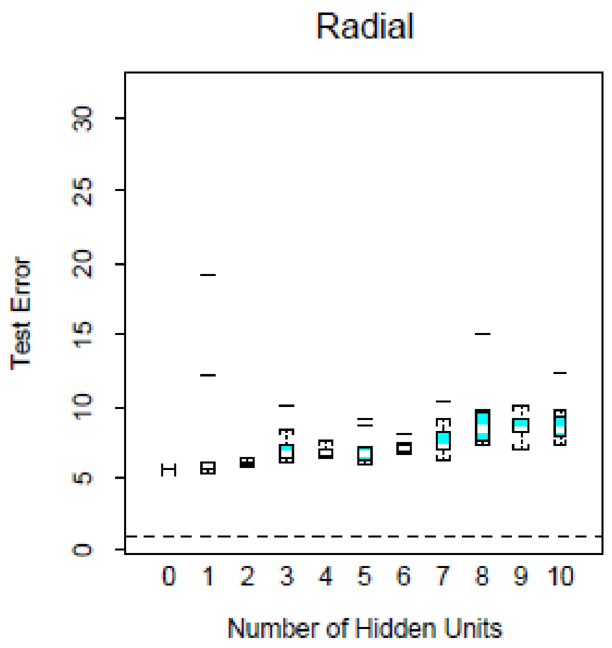
Both the approach to select a high number of functions and optimize the weight decay and the approach to fix the weight decay and optimize the number of hidden units gives good result

Examples of simulated data

$$\text{Radial: } Y = \prod_{m=1}^{10} \phi(X_m) + \varepsilon_2. \qquad \phi(t) = (1/2\pi)^{1/2} \exp(-t^2/2)$$

- $\frac{\mathrm{Var}(f(X))}{\mathrm{Var}(\varepsilon)} \approx 4$
- Training data size: 100 samples
- Test data size : 10 000 samples

NN does not always work:
All cases are worse than the mean
And it gets even worse as the number of units increase



Alternative models for neural networks

11.7 Example: ZIP Code Data 405

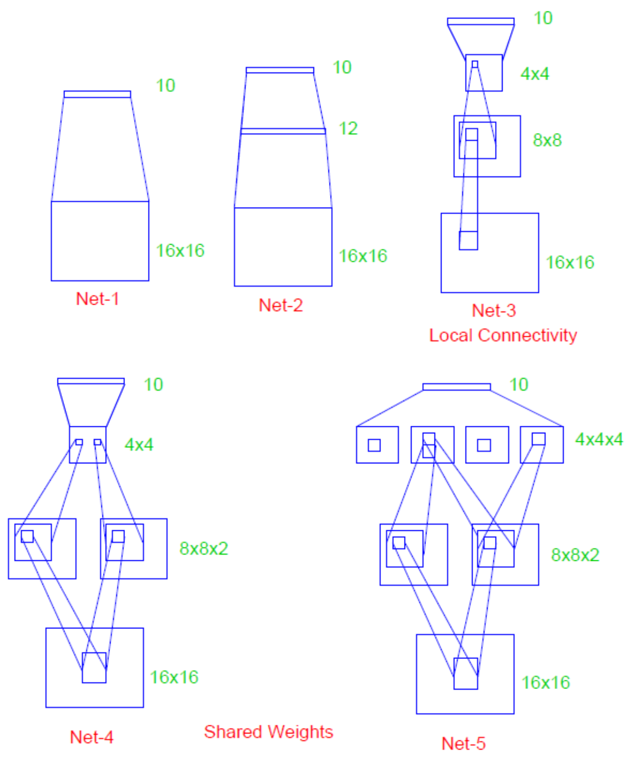


FIGURE 11.10. *Architecture of the five networks used in the ZIP code example.*

TABLE 11.1. *Test set performance of five different neural networks on a hand-written digit classification example (Le Cun, 1989).*

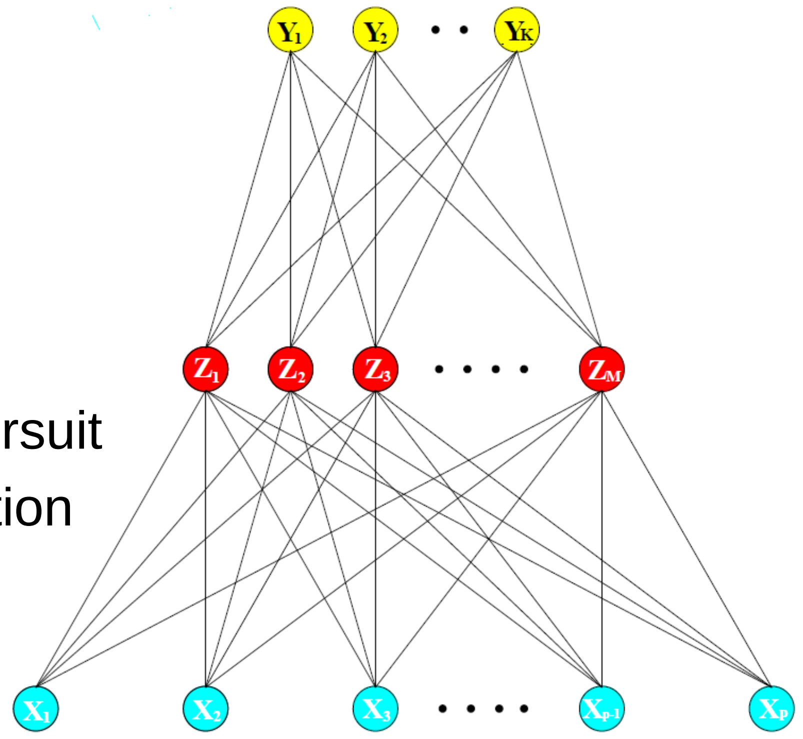
	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

«Hand crafting» NN might help
By reducing the number of
parameters to be estimated.

- Setting weights to zero (localize)
- Setting weights equal
  (Convolutional NN)

Learning today Neural nets

- Projection pursuit
  - What is it?
  - How to solve it: Stagewise
- Neural nets
  - What is it?
  - Graphical display
  - Connection to Projection pursuit
  - How to solve it: Backprojection
  - Stochastic Gradient search
  - Deep and wide



Exercise Projection pursuit

- Derive the expressions for w which minimizes the linearize expression for the object function:

$$\sum_{i=1}^{n} g'\left(w_{\text{old}}^T x_i\right)^2 \left(\frac{y_i - g\left(w_{\text{old}}^T x_i\right)}{g'\left(w_{\text{old}}^T x_i\right)} + w_{\text{old}}^T x_i - w^T x_i\right)^2$$

- Hint introduce the quantities below, and look up weighted regression
  - $a_i = g'\left(w_{\text{old}}^T x_i\right)^2$ with $a = [a_1, \dots, a_N]^T$
  - $b_i = \frac{y_i - g\left(w_{\text{old}}^T x_i\right)}{g'\left(w_{\text{old}}^T x_i\right)} + w_{\text{old}}^T x_i$ with $b = [b_1, \dots, b_N]^T$

Exercise continue.

- #11.3
- #NN: Recreate figure 11.17 (left) using the `neuralnet` function in library «neuralnet» in R.
  - Use the function `calculate` to get the prediction for the test data
- #PP: Use the `ppr` function in library «stats» in R to create a similar plot for projection pursuit.
  - Use the function `predict` to get the prediction for the test data
- Extra:
  - Do the same for the function in exercise 11.5