Cracow School of Theoretical Physics, May 30 – June 8, 2003, Zakopane

Neural networks

and the algebra of gradients

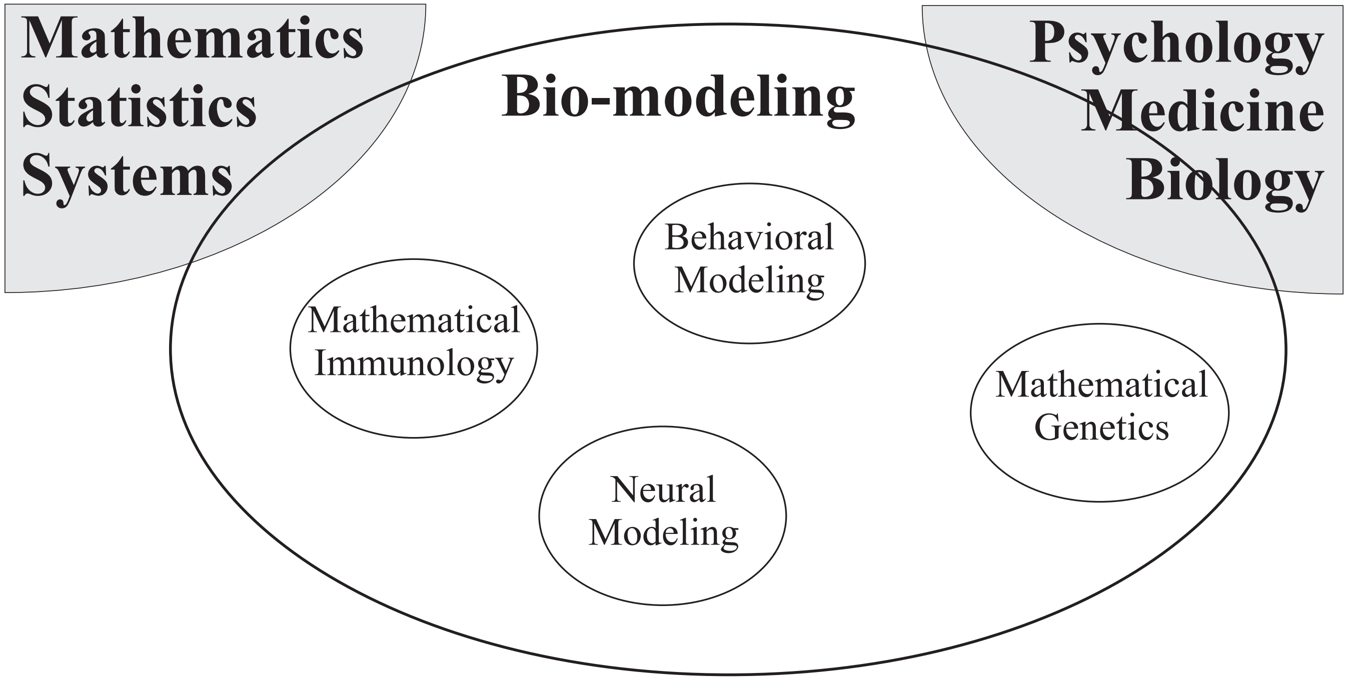
Andrzej Pacut

Warsaw University of Technology
Nowowiejska 15/19, 00–665 Warsaw

pacut@ia.pw.edu.pl
http://www.ia.pw.edu.pl/~pacut

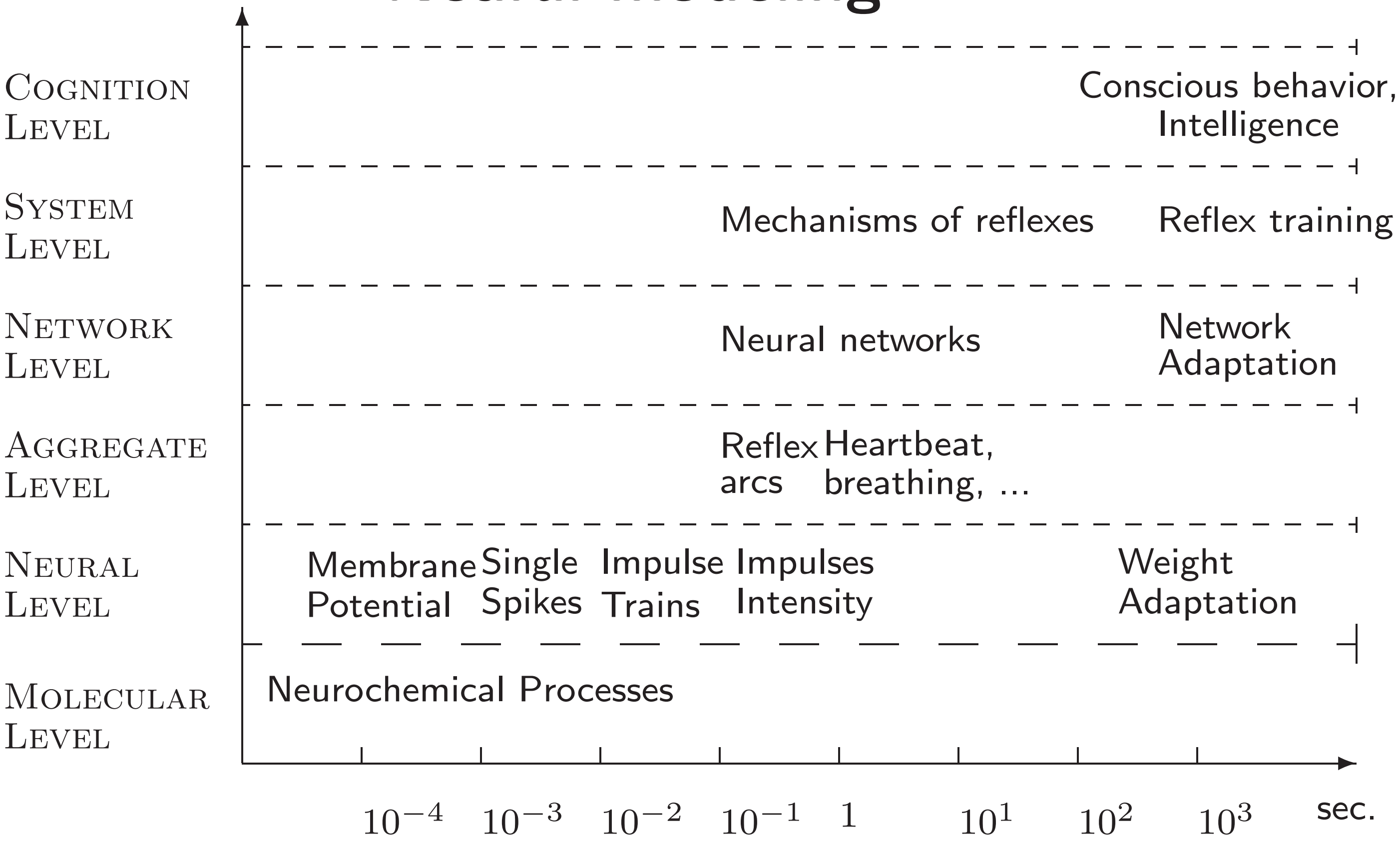
Neural networks and the algebra of gradients

Bio-modeling

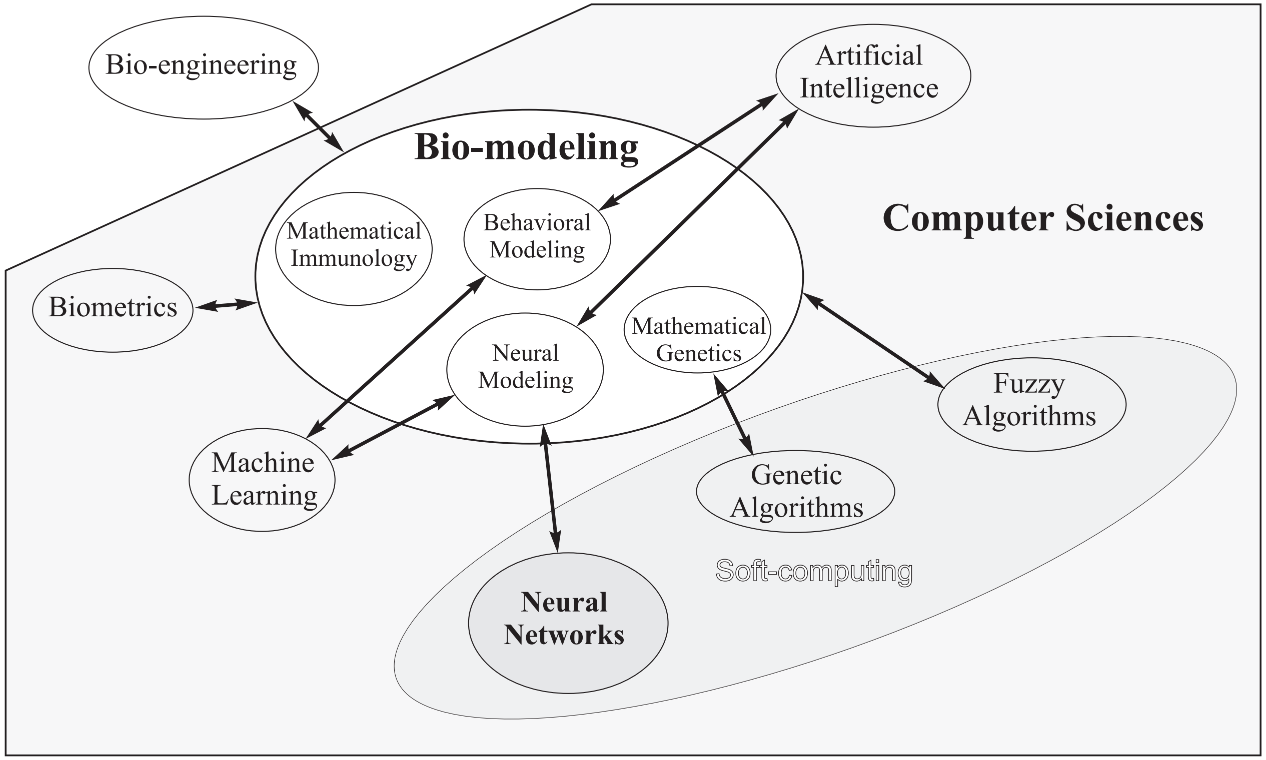


- goals: help to understand
- applications: biology, medicine, pharmacy, psychology

Neural modeling



Biological inspirations



- goals: application-specific (*technical*)
- cutting-off biological roots; *the results do not copy biology*
- need of recurrent inspiration

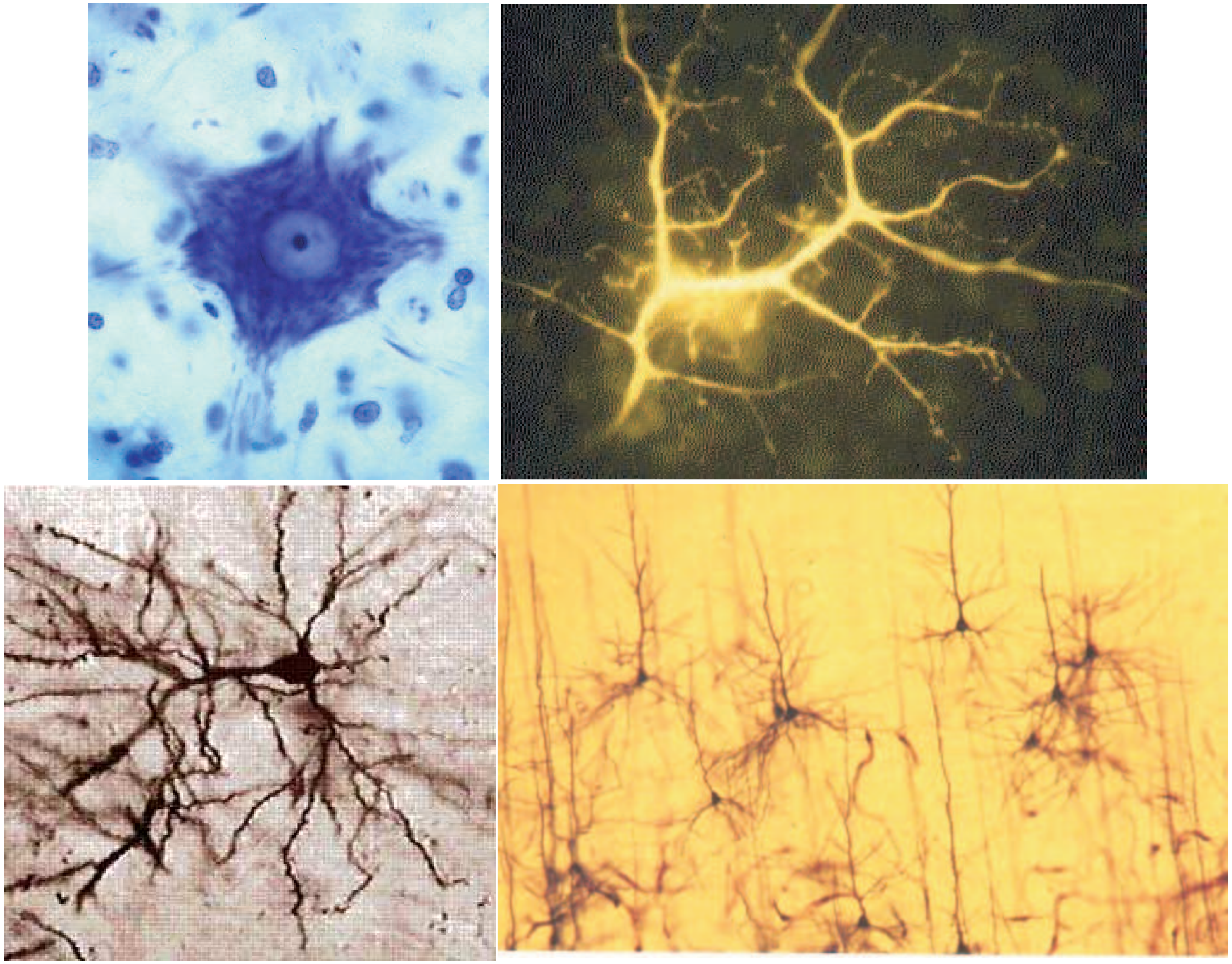
Milestones

- **McCulloch i Pitts 1943** ▶ *model of neuron* $y(t+1) = \mathbf{1}(\mathbf{w}^T\mathbf{u} - \vartheta)$
- **Rosenblatt 1958** (Cornell U., Ithaca, NY) ▶ *perceptron*
- Widrow 1960 (Stanford U., Stanford, CA) ▶ *Adaline*
- **Minsky 1969** (MIT, Cambridge, MA) ▶ *perceptron + artificial intelligence*
- “quiet decade”
  - Kohonen 1972 (Helsinki U. of Techn.) ▶ *linear associative memory*
  - Werbos 1974 (Harvard U. Ph.D., Cambridge, MA) ▶ *backpropagation*
  - Grossberg 1983 (Boston U., Boston, MA) ▶ *stability of dynamic networks*
- **Hopfield 1982** (Princeton U., Princeton, NJ) ▶ *memory – network stability*
- **Rummelhart, Hinton, Williams 1986** (PDP group, MIT, Cambridge, MA) ▶ *backpropagation learning*

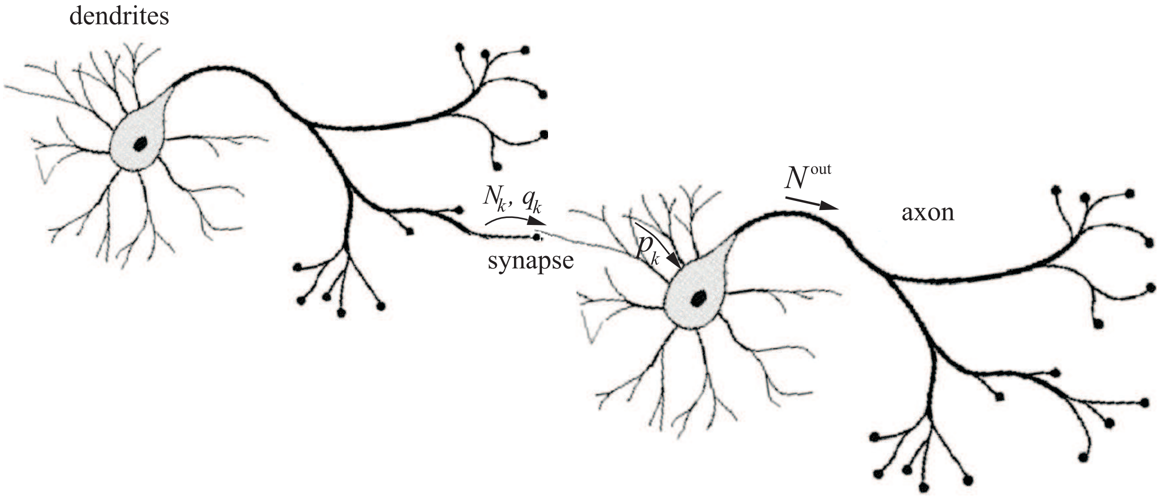
Artificial neural networks — rough characterization

- modularity
- parallelism
- nonlinearity
- plasticity
- fault tolerance
- over-parametrization

Neurons

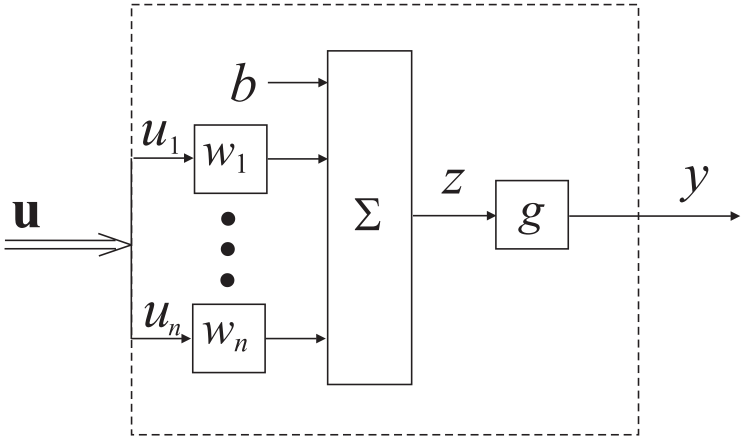


Typical neuron



- *cell body* diameter $5\mu m$ (granular cells) – 100 μm (motor neurons)
- *nucleus*, *RNA*, *protein-manufacturing structures*
- *cell membrane* 5 nm separates cytoplasm from extracellular fluid; controlled diffusion (*gating*) and active passage (*pumping*) of chemicals
- *axon*: output projection, $1 - 1000$ mm, from *axon hillock*, branches into *collaterals* towards its end, and terminates at *synapses* of other cells; carries *action potential*
- *dendrites*: input projections, up to 0.1 mm; receive and integrate signals
- *synapses* contact points, several to 90 tys. (Purkinje cells) per axon; human: 10^{15}

ANN neuron

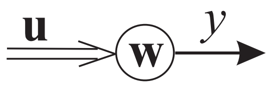


linear transformation

$$z = b + \sum_{i=1}^{n} w_i\, u_i = \sum_{i=0}^{n} w_i\, u_i$$

activation function

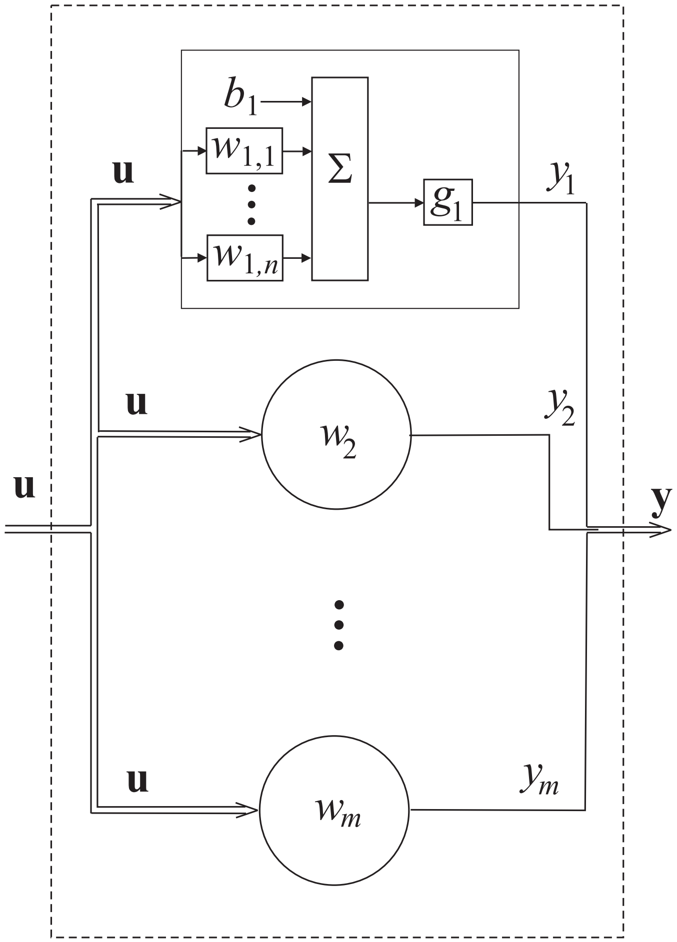
$$y = g(z)$$



neuron's equation

$$y = g\Big(b + \sum_{i=1}^{n} w_i\, u_i\Big) = g\Big(\sum_{i=0}^{n} w_i\, u_i\Big)$$

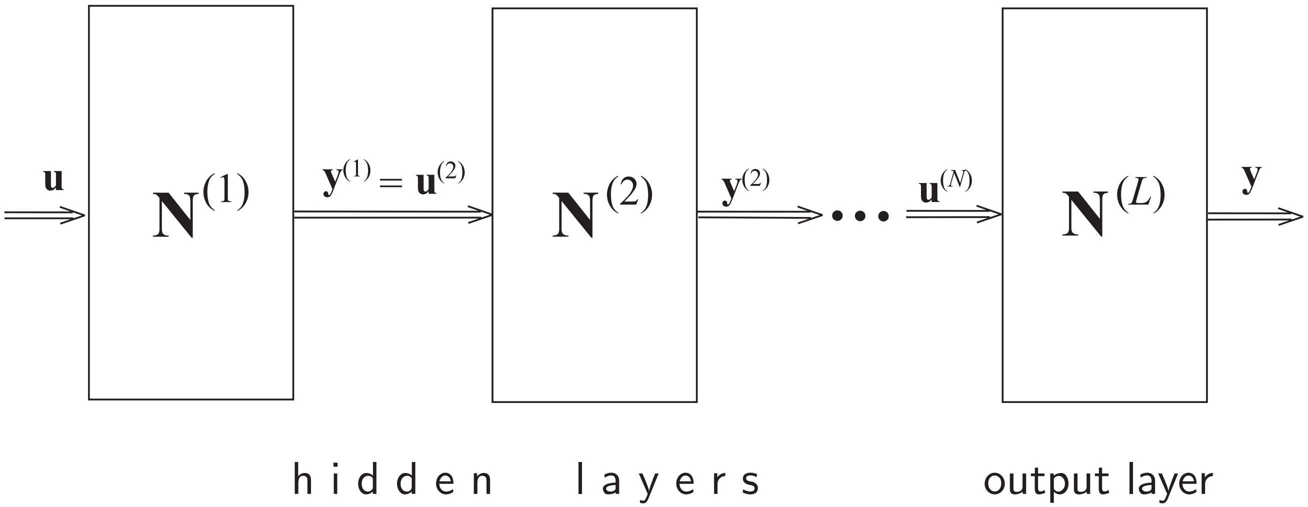
Layer



$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \mathbf{g}(\mathbf{W}\mathbf{u}), \qquad \mathbf{g}(\mathbf{z}) = \begin{bmatrix} g_1(z_1) \\ g_2(z_2) \\ \vdots \\ g_m(z_m) \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_m^T \end{bmatrix} = \begin{bmatrix} b_1 & w_{1,1} & \dots & w_{1,n} \\ b_2 & w_{2,1} & \dots & w_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ b_m & w_{m,1} & \dots & w_{m,n} \end{bmatrix}$$

Multi-layer perceptron



$$\mathbf{u}^{(1)} = \mathbf{u}, \qquad \mathbf{u}^{(\ell)} = \mathbf{y}^{(\ell-1)}, \quad \ell = 2, \ldots, L$$

$$\mathbf{y}^{(\ell)} = \mathbf{N}^{(\ell)}(\mathbf{u}^{(\ell)}), \quad \ell = 1, \ldots, L$$

$$\mathbf{y} = \mathbf{y}^{(L)}$$

Applications of ANN

- *function approximation*
- *clustering*
- *global optimization*

- classification and pattern recognition (speech, writing)
- control, robotics, expert systems
- adaptive filtering, DSP, prediction
- military applications: recognition, target identification
- economics, management, decision making
- biometrics
- biology (DNA modeling)
- telecommunication
- industry

Approximation problem

- approximation of unknown function $f \in \mathcal{F}$
  select $\widehat{f}$ from a family $\widehat{\mathcal{F}}$ to minimize $d(f, \widehat{f})$
- $\mathcal{F}$ results from applications ($\mathcal{C}(\mathcal{U})$, $\mathcal{C}_k(\mathcal{U})$, $\mathcal{L}_p(\mathcal{U})$)
- d – metric for $\mathcal{F}$
- unknown function, samples accessible

$$\big(\mathbf{u}(1), \mathbf{y}(1)\big),\ \big(\mathbf{u}(2), \mathbf{y}(2)\big),\ \ldots,\ \big(\mathbf{u}(N), \mathbf{y}(N)\big)$$

- is $\widehat{\mathcal{F}}$ sufficiently rich?
- *universal approximator*
  for arbitrary $\epsilon > 0$, $f \in \mathcal{F}$ there exists $\widehat{f} \in \widehat{\mathcal{F}}$ for which

$$d(f, \widehat{f}) < \epsilon$$

- **are NN universal approximators ?**

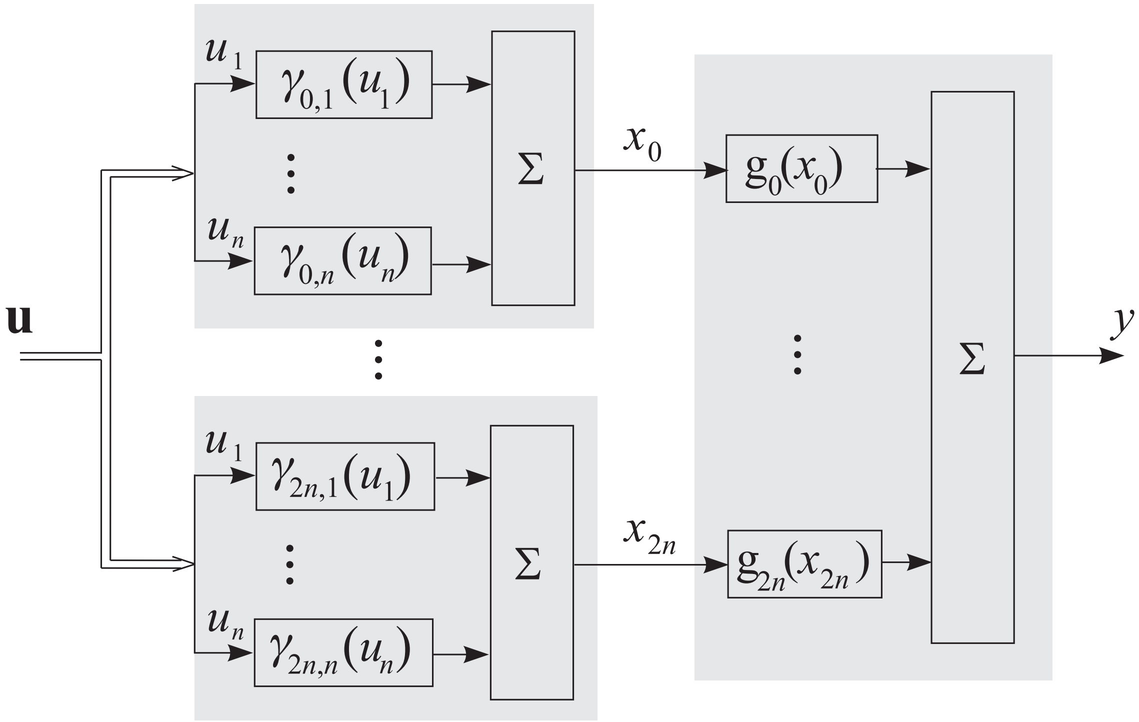
Kolmogorov's representation (1)

- [13 HYPOTHESIS, HILBERT] there exists an analytic function of 3 variables that *is not* a finite superposition of continuous functions of 2 variables
- KOLMOGOROV every continuous function of n arguments $f : \mathcal{I}^n \mapsto \mathbb{R}$, $n \geq 2$, can be presented in the form

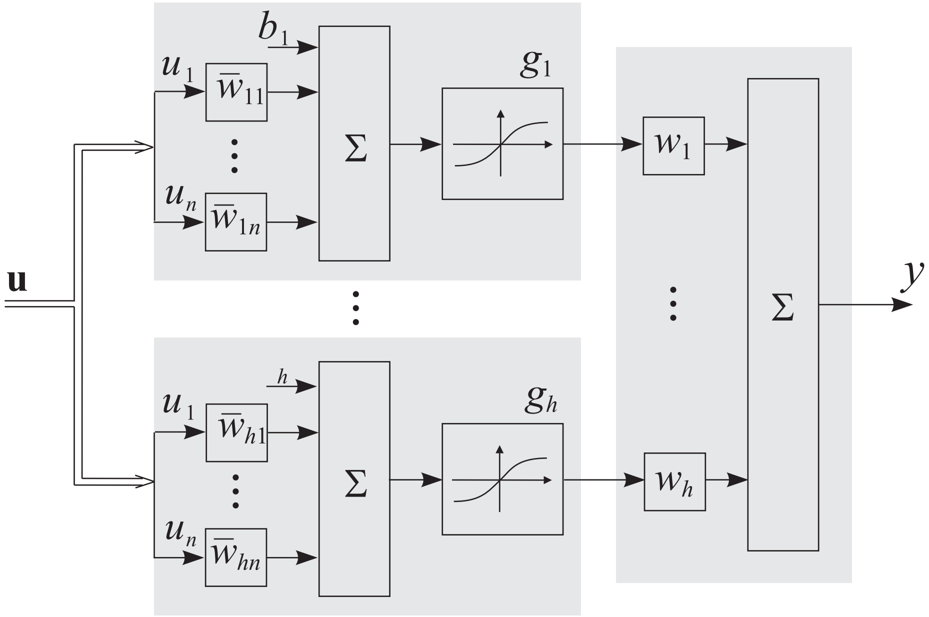
$$f(\mathbf{u}) = \sum_{i=0}^{2n} g_i \left(\sum_{j=1}^{n} \gamma_{i,j}(u_j) \right), \qquad \mathbf{u} = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \in \mathcal{I}^n$$

  - $\{g_0, \ldots g_{2n}\}$ are continuous functions of 1 variable and depend on f
  - a family of monotonic functions $\gamma_{i,j} : \mathcal{I} \mapsto \mathbb{R}$ is universal

Kolmogorov's representation (2)



Standard approximating network



- approximation of $f : \mathcal{U} \subset \mathbb{R}^n \mapsto \mathbb{R}$
- family of two-layer networks $\mathcal{N}[g]$
  - linear output layer with zero bias
  - hidden layer of identical activation functions g
- "free" parameters:
  - hidden layer size h
  - nh weights and h biases of hidden layer
  - h weights of the output layer

Approximation of continuous functions

- approximating functions are *continuous on a compact domain* $\mathcal{U} \subset \mathbb{R}^n$
- distance induced by the sup norm

$$d(f, \widehat{f}) = \sup_{\mathbf{u} \in \mathcal{U}} |f(\mathbf{u}) - \widehat{f}(\mathbf{u})|$$

- [Hornik, 1991] if the activation function g is

  continuous, *bounded* and *non-constant*

  then $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well (in a supremum sense) approximate any continuous function on a compact domain

- i.e. for any f continuous on compact domain $\mathcal{U}$ and arbitrary $\epsilon > 0$, there exist $\mathcal{N}[g]$ network such that

$$\sup_{\mathbf{u} \in \mathcal{U}} |f(\mathbf{u}) - \widehat{f}(\mathbf{u})| < \epsilon$$

Approximation of continuous functions - necessary and sufficient condition

- $g : \mathbb{R} \mapsto \mathbb{R}$ is *locally essentially bounded*, if it is *bounded a.e.* on every compact domain

- [Leshno, 1993] if
  - activation function g is locally essentially bounded
  - closure of the discontinuity set of g has Lebesgue measure zero

  then $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well approximate (in the sup sense) any continuous function **if and only if** g *is not a polynomial a.e.*

Approximation of functions and their derivatives

- approximated functions: $\mathcal{C}_m(\mathcal{U})$ on a compact domain $\mathcal{U} \in \mathbb{R}^n$
- [Hornik, Stinchcombe, White, 1990] if the activation function g is

  m-times continuously differentiable, bounded and non constant

  then $\mathcal{N}[g]$ with sufficiently large hidden layer can arbitrarily well approximate (in the sup sense) any continuous function together with its derivatives up to order m
- i.e., for any function in $\mathcal{C}_m(\mathcal{U})$ on a compact domain $\mathcal{U} \in \mathbb{R}^n$ and every $\epsilon > 0$ there exists $\mathcal{N}[g]$ for which

$$\sup_{\mathbf{u} \in \mathcal{U}} |D^{\mathbf{k}} f(\mathbf{u}) - D^{\mathbf{k}} \widehat{f}(\mathbf{u})| \leq \epsilon$$

for all derivatives $D^{\mathbf{k}} = \frac{\partial^{k_1 + \cdots + k_n}}{\partial u_1^{k_1} \ldots \partial u_n^{k_n}}$

Approximation of integrable functions

- approximated functions $f \in \mathcal{L}_p$, $1 \leq p < \infty$, on a bounded domain $\mathcal{U} \in \mathbb{R}^n$
- distance induced by the norm

$$\|f\|_p = \Big(\int_{\mathcal{U}} |f(\mathbf{u})|^p \, d\mathbf{u}\Big)^{1/p}$$

- [Hornik, 1991] if the activation function g is

  bounded and *non-constant*

  then $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well (in $\|f\|_p$ norm) approximate any function in $\mathcal{L}_p$ on a compact domain
- i.e. for any function in $\mathcal{L}_p$ on a compact set $\mathcal{U} \in \mathbb{R}^n$ and any $\epsilon > 0$ there exists $\mathcal{N}[g]$ for which

$$\|f - \widehat{f}\|_p < \epsilon$$

Approximation of integrable functions - necessary and sufficient condition

- [Leshno, 1993] if
  - activation function g is locally essentially bounded
  - closure of the discontinuity set of g has Lebesgue measure zero

  then $\mathcal{N}[g]$ of sufficiently large hidden layer can arbitrarily well approximate any $\mathcal{L}_p$ function **if and only if** g *is not a polynomial a.e.*

Approximation of integrable functions and their derivatives

- approximated functions together with their derivatives up to mth order are

  continuous

  integrable with p-th power

- [Hornik, Stinchcombe, White, 1990] if the activation function g is

  m-times continuously differentiable, *bounded* and *non-constant*

  then a $\mathcal{N}[g]$ network of sufficiently large hidden layer can arbitrarily well approximate (in $\|f\|_p$ sense) any function together with its derivatives up to order m

- i.e., for any function whose derivatives up to order m are continuous and p-times integrable ($m \geq 1$, $1 \leq p < \infty$) and arbitrary $\epsilon > 0$ there exists a $\mathcal{N}[g]$ network for which

$$\|D^{\mathbf{k}} f - D^{\mathbf{k}} \widehat{f}\|_p < \epsilon$$

  for any $\mathbf{k} : |\mathbf{k}| \leq m$

Approximation of measurable functions

- approximated functions: measurable on a compact domain
- [Luzin] any measurable function can be modified to a continuous function on a set of arbitrarily small measure
- if the activation function g is

  continuous, bounded and non-constant

  then a $\mathcal{N}[g]$ network of sufficiently large hidden layer can arbitrarily well approximate (in the sup sense) *almost everywhere* any measurable function on compact domain
- i.e., for any every measurable function on a compact domain and arbitrary $\epsilon > 0$ there exist a $\mathcal{N}[g]$ network and a compact set $\mathcal{U}_\epsilon \subset \mathcal{U}$ arbitrarily close to $\mathcal{U}$ in a sense that $P(\mathcal{U} - \mathcal{U}_\epsilon) \leq \epsilon$, for which

$$\sup_{\mathbf{u} \in \mathcal{U}_\epsilon} |f(\mathbf{u}) - \widehat{f}(\mathbf{u})| \leq \epsilon$$

Approximation of random functions

- $\mathbf{u}$ is a random variable defined on $\mathcal{U} \subset \mathbb{R}^n$
- approximated functions: p-th order functions of $\mathbf{u}$, i.e. $\mathcal{E}|f(\mathbf{u})|^p < \infty$ for a fixed p, $1 \le p < \infty$
- function distance defined in the sense of p-th moments, namely $d(f, \widehat{f}) = \left(\mathcal{E}\left(f(\mathbf{u}) - \widehat{f}(\mathbf{u})\right)^p\right)^{1/p}$
- [Hornik, 1991] if the activation function g is

  bounded and *non-constant*

  then a $\mathcal{N}[g]$ network of sufficiently large hidden layer can arbitrarily well approximate (in the sense of p-th moments) any p-th order function of $\mathbf{u}$
- i.e., for any f satisfying $\mathcal{E}|f(\mathbf{u})|^p < \infty$ and arbitrary $\epsilon > 0$ there exists a $\mathcal{N}[g]$ network for which

$$\mathcal{E}|f(\mathbf{u}) - \widehat{f}(\mathbf{u})|^p < \epsilon$$

Approximation of random functions – necessary and sufficient condition

- [Leshno, 1993] if
  - the activation function g is locally essentially bounded
  - the closure of discontinuity points of g has the Lebesgue measure zero

  then a $\mathcal{N}[g]$ network of sufficiently large hidden layer can arbitrarily well approximate any p-th order function of a continuous random variable $\mathbf{u}$ **if and only if** g *is not a polynomial a.e.*

Approximation of a random function and its derivatives

- $\mathbf{u}$ is a continuous random variable defined on a compact set $\mathcal{U} \subset \mathbb{R}^n$
- approximated functions:

  continuous with derivatives up to order m

  f and its derivatives $D^{\mathbf{k}} f(\mathcal{U})$ are random variables of order p, $1 \leq p < \infty$
- [Hornik, Stinchcombe, White, 1990] if the activation function g is

  m-times continuously differentiable, bounded and non-constant

  then a $\mathcal{N}[g]$ network of sufficiently large hidden layer can arbitrarily well (in the sense of p-th moments) approximate any continuous function and its derivatives up to order m
- i.e., for any m-times continuously differentiable function of a random variable of order p together with its derivatives, and arbitrary $\epsilon > 0$ there exist a $\mathcal{N}[g]$ network for which

$$\mathcal{E}|D^{\mathbf{k}}\big(f(\mathbf{u}) - \widehat{f}(\mathbf{u})\big)|^p < \epsilon$$

No curse of dimensionality ?

- $\mathcal{N}[g]$ where g is a bounded continuous sigmoid, $f : \mathbb{R}^n \mapsto \mathbb{R}$ is approximated for $\|\mathbf{u}\| \leq r$, $P\{\|\mathbf{u}\| \leq r\} = 1$, approximation error $d(f, \widehat{f}) = \left(\mathcal{E}\, |f - \widehat{f}|^2\right)^{1/2}$
- function complexity index

$$C_f = \int \|\boldsymbol{\omega}\|^2 \, |\bar{f}(\boldsymbol{\omega})| \, d\boldsymbol{\omega}$$

  where $\bar{f}(\boldsymbol{\omega}) = \int \exp(j\, \boldsymbol{\omega}^T \mathbf{u})\, f(\mathbf{u})\, d\mathbf{u}$, $\boldsymbol{\omega} \in \mathbb{R}^n$ denotes Fourier transform of f
- [Barron, 1993] if $C_f < \infty$ then there exists a $\mathcal{N}[g]$ network with h hidden neurons such that the approximation error is bounded by

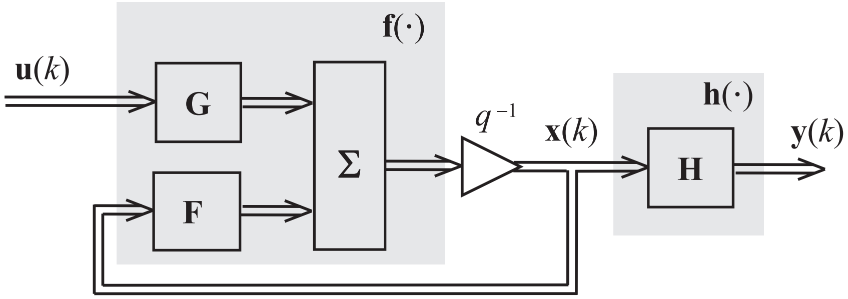
$$\varepsilon \leq \frac{2\, r\, C_f}{\sqrt{h}}$$

- hidden layer size bound to keep $\varepsilon < \varepsilon_0$ does not depend on the input dimension n

$$h \leq \frac{4\, r^2\, C_f^2}{\varepsilon_0^2}$$

- for polynomial approximations, splines, etc., the required number of parameters grows exponentially with the input space dimension
- the bound depends on the function complexity index C_f that may depend on n

ARX representation of linear systems



$$\mathbf{x}(t{+}1) = \mathbf{F}\,\mathbf{x}(t) + \mathbf{G}\,\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{H}\,\mathbf{x}(t)$$

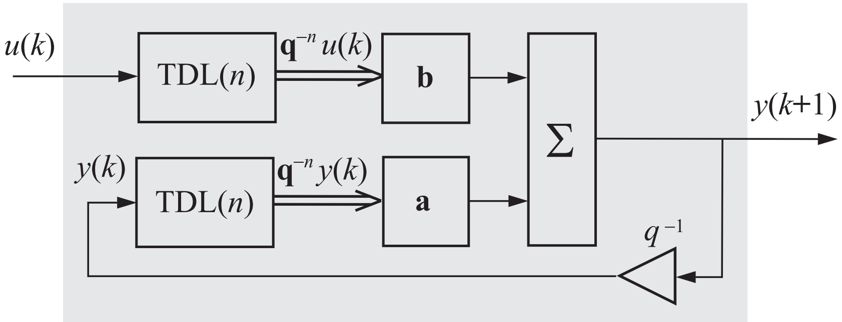
- if the system is *observable* then the state is linearly dependent on at most n inputs and outputs - ARX(n) representation

$$\mathbf{y}(t+1) = \sum_{i=1}^{n} \mathbf{A}_i\,\mathbf{y}(t-i+1) + \sum_{i=1}^{n} \mathbf{B}_i\,\mathbf{u}(t-i+1)$$

$$\mathbf{y}(t{+}1) = \mathbf{A}\,\mathbf{q}^{-n}\mathbf{y}(t) + \mathbf{B}\,\mathbf{q}^{-n}\mathbf{u}(t)$$

for SISO systems

$$y(t{+}1) = \mathbf{a}^T\mathbf{q}^{-n}y(t) + \mathbf{b}^T\mathbf{q}^{-n}u(t)$$



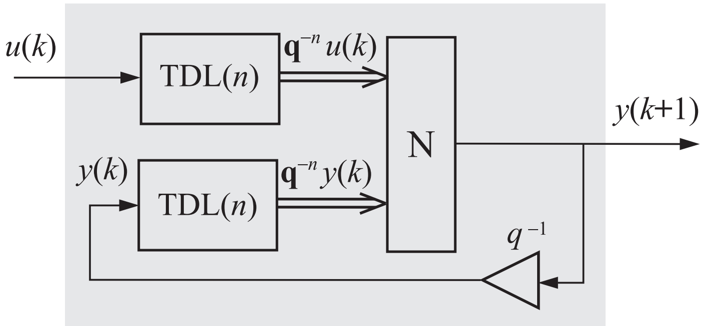
NARX representation for nonlinear systems

- if a *linearized system* is observable then **locally around the equilibrium** the state can be eliminated from the state equations, yet the relations **are not linear**: *N(onlinear)ARX*

$$\mathbf{y}(t+1) = \psi\big(\mathbf{q}^{-n}\mathbf{y}(t),\, \mathbf{q}^{-n}\mathbf{u}(t)\big)$$

- **global representations are unknown** - except in special cases:
  - *autonomous systems*
  - *if* $\mathbf{f}$, $\mathbf{g}$ *are sufficiently smooth and* $\mathbf{f}$ *is state invertible*

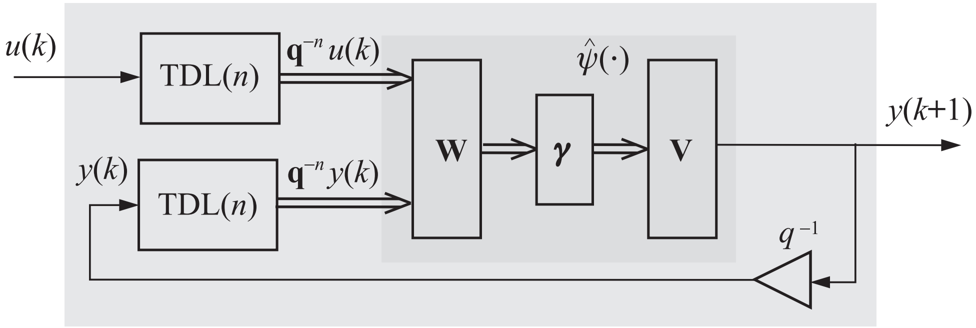
  then $2n+1$ delayed outputs (and inputs) suffice to eliminate the state



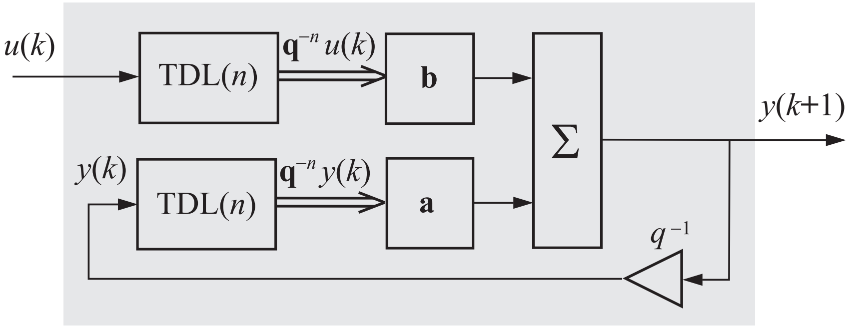
Neural realization of NARX

- nonlinear functions can be approximated by neural networks: *N(eural)ARX*

$$\mathbf{y}(t+1) = \widehat{\boldsymbol{\psi}}\big(\mathbf{q}^{-n}\mathbf{y}(t),\, \mathbf{q}^{-n}\mathbf{u}(t);\, \mathbf{w}\big)$$



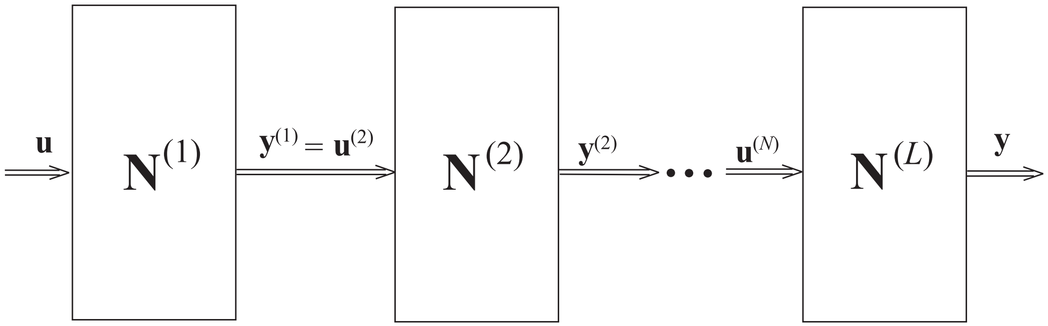
- ARX corresponds to NARX without the hidden layer



Layer system

$$
\begin{array}{ll}
f_1 : X_0 \mapsto X_1, & x_1 = f_1(x_0) \\
\quad \dots & \\
f_k : X_{k-1} \mapsto X_k, & x_k = f_k(x_{k-1}) \\
\quad \dots & \\
f_n : X_{n-1} \mapsto X_n & x_n = f_n(x_{n-1})
\end{array}
$$

$$x_n = f_n(f_{n-1}(\dots(f_1(x_0))))$$



Derivatives for layer system

- notation

$$f'_k = \frac{df_k}{dx_{k-1}} \qquad \text{for } i = 1, \ldots, n$$

$$x'_{k|\ell} = \frac{dx_k}{dx_\ell} \qquad \text{for fixed } x_0, \ldots x_{\ell-1}, \quad \ell < k$$

- derivative

$$\frac{dx_n}{dx_0} = f'_n(x_{n-1})\, f'_{n-1}(x_{n-2}) \ \ldots\ f'_2(x_1)\, f'_1(x_0)$$

with

$$x_k = f_k(f_{k-1} \cdots f_1(x_0)), \quad k = 1, \ldots, n$$

Chain rule — forward propagation

- "output to input" grouping

$$x'_{n|0} = \frac{dx_n}{dx_0} = \underbrace{f'_n(x_{n-1})\ f'_{n-1}(x_{n-2})\ \ldots\ \underbrace{f'_2(x_1)\ \underbrace{f'_1(x_0)}_{x'_{1|0}}}_{x'_{2|0}}}_{x'_{n-1|0}}$$

- "input to output" gradient calculations

$$\begin{aligned} x'_{0|0} &= 1 \\ x'_{k|0} &= f'_k(x_{k-1})\ x'_{k-1|0} \\ x_k &= f_k(x_{k-1}) \qquad k = 1, \ldots, n \end{aligned}$$

- derivatives and function values calculated in single pass, for $k = 1, \ldots, n$

Backpropagation

- "input to output" grouping

$$x'_{n|0} = \frac{dx_n}{dx_0} = \underbrace{\underbrace{\underbrace{f'_n(x_{n-1})}_{x'_{n|n-1}} f'_{n-1}(x_{n-2})}_{x'_{n|n-2}} \dots f'_2(x_1)}_{x'_{n|1}} f'_1(x_0)$$

- "output to input" derivatives calculations
  - forward pass: function values

$$x_k = f_k(x_{k-1}), \qquad k = 1, \dots, n$$

  - backward pass: derivatives

$$\begin{aligned} x'_{n|n} &= 1 \\ x'_{n|\ell} &= x'_{n|\ell+1}\, f'_{\ell+1}(x_\ell), \qquad \ell = n-1, \dots, 0 \end{aligned}$$

Illustration (one-dimensional)

$$y = \sin(\exp(x^3 \cos(x^3)))$$

layer system

$(x_0 = x,\ x_4 = y)$

$$x_1 = x_0^3$$
$$x_2 = x_1 \cos(x_1)$$
$$x_3 = \exp(x_2)$$
$$x_4 = \sin(x_3)$$

Forward propagation

$$x'_{4|0} = \cos(x_3)\, x'_{3|0}$$
$$x'_{3|0} = \exp(x_2)\, x'_{2|0}$$
$$x'_{2|0} = \big(\cos(x_1) - x_1 \sin(x_1)\big)\, x'_{1|0}$$
$$x'_{1|0} = 3\, x_0^2\, x'_{0|0}$$
$$x'_{0|0} = 1$$

Backpropagation

$$x'_{4|0} = 3\, x'_{4|1}\, x_0^2$$
$$x'_{4|1} = \big(\cos(x_1) - x_1 \sin(x_1)\big)\, x'_{4|2}$$
$$x'_{4|2} = x'_{4|3} \exp(x_2)$$
$$x'_{4|3} = x'_{4|4} \cos(x_3)$$
$$x'_{4|4} = 1$$

Illustration (multi-dimensional)

$$y = \sin\bigl(\exp(x^3) + 4\cos(x^3)\bigr)$$

layer system
($x_0 = x$, $x_3 = y$)

$$x_1 = x^3$$
$$x_{21} = \exp(x_1)$$
$$x_{22} = \cos(x_1)$$
$$x_3 = \sin(x_{21} + 4x_{22})$$

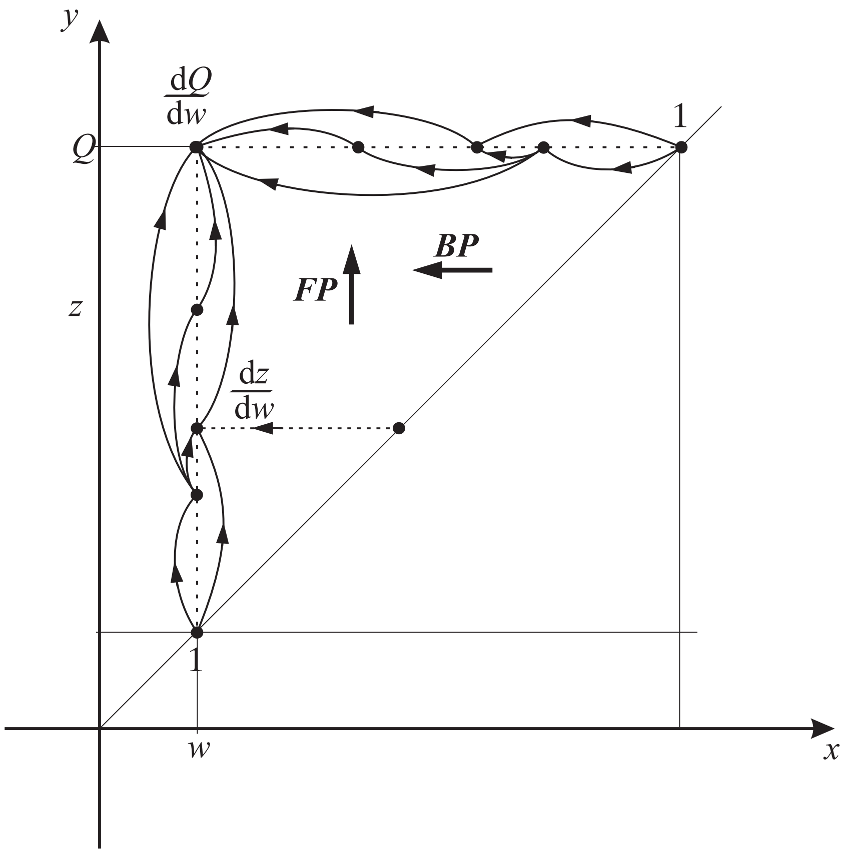
forward propagation

$$x'_{3|0} = \cos(x_{21}+4x_{22})\bigl(x'_{21|0} + 4x'_{22|0}\bigr)$$
$$x'_{21|0} = \exp(x_1)\, x'_{1|0}$$
$$x'_{22|0} = -\sin(x_1)\, x'_{1|0}$$
$$x'_{1|0} = 3\,x^2$$

backpropagation

$$x'_{3|0} = 3\, x'_{3|1}\, x^2$$
$$x'_{3|1} = x'_{3|22}\, \exp(x_1) - x'_{3|21} \sin(x_1)$$
$$x'_{3|21} = \cos(x_{21} + x_{22})$$
$$x'_{3|22} = 4\, \cos(x_{21} + x_{22})$$

Comparison



- forward-propagation (FP): derivatives of intermediate variables *w.r. to the target independent variable*
- backpropagation (BP): derivatives *of the target dependent variable* w.r. to intermediate variables
- neural network: required gradient of *cost* w.r. to the weights

Ordered system

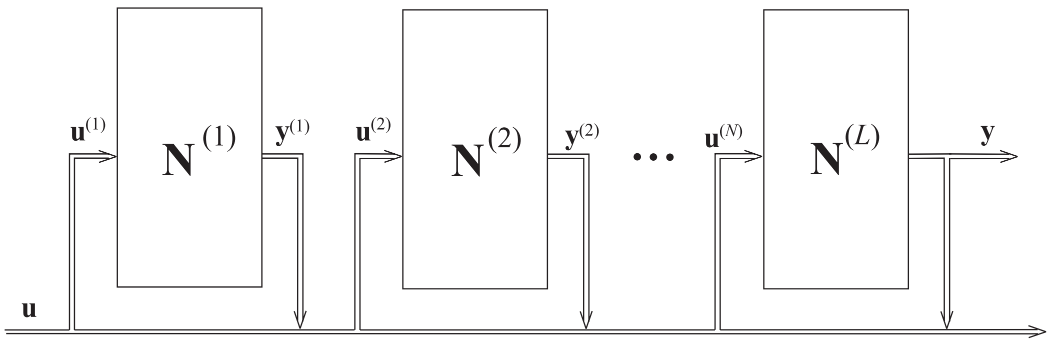
$$f_1 : X_0 \mapsto X_1 \qquad\qquad x_1 = f_1(x_0)$$

$$\dots$$

$$f_k : X_0 \times \cdots \times X_{k-1} \mapsto X_k \qquad\qquad x_k = f_k(x_0, \dots, x_{k-1})$$

$$\dots$$

$$f_n : X_0 \times \cdots\cdots \times X_{n-1} \mapsto X_n \qquad\qquad x_n = f_n(x_0, \dots\dots, x_{n-1})$$



Gradients for ordered system

- denote (here $x_k \in \mathbb{R}$)

$$f'_{k|\ell}(x_0, \ldots, x_{k-1}) = \frac{\partial}{\partial x_\ell} f_k(x_0, \ldots, x_{k-1}), \quad 0 \le \ell < k$$

- for the first ℓ variables $x_0, \ldots, x_{\ell-1}$, $\ell < k$, fixed, define

$$x'_{k|\ell} = \frac{dx_k}{dx_\ell}$$

- the following equations are equivalent

$$x'_{k|\ell} = f'_{k|\ell} + \sum_{i=\ell+1}^{k-1} f'_{k|i}\, x'_{i|\ell} \quad 0 \le \ell < k \le n \tag{FP}$$

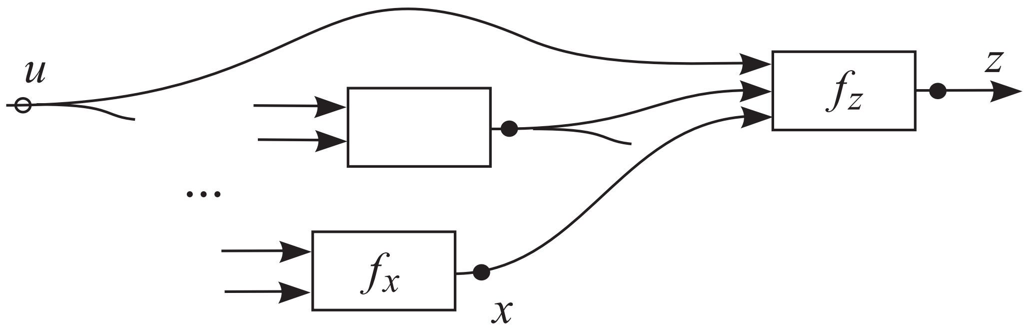
$$x'_{k|\ell} = f'_{k|\ell} + \sum_{j=\ell+1}^{k-1} x'_{k|j}\, f'_{j|\ell} \quad 0 \le \ell < k \le n \tag{BP}$$

Forward propagation - local form

- for any two $u = x_k$, $z = x_\ell$, $\ell < k$

$$\frac{dz}{du} = \frac{\partial f_z}{\partial u} + \sum_x \frac{\partial f_z}{\partial x} \frac{dx}{du}$$

where $f_z = f_\ell$ is a function that defines z, and summation extends over all x *that directly influence* z (i.e., arguments of f_z)



Backpropagation - local form

- for any two variables $u = x_k$, $z = x_\ell$, $\ell < k$

$$\frac{dz}{du} = \frac{\partial f_z}{\partial u} + \sum_x \frac{dz}{dx}\frac{df_x}{du}$$

where f_x is a function that defines x, and summation extends over all x *directly influenced by* u (all functions for which u is one of arguments)

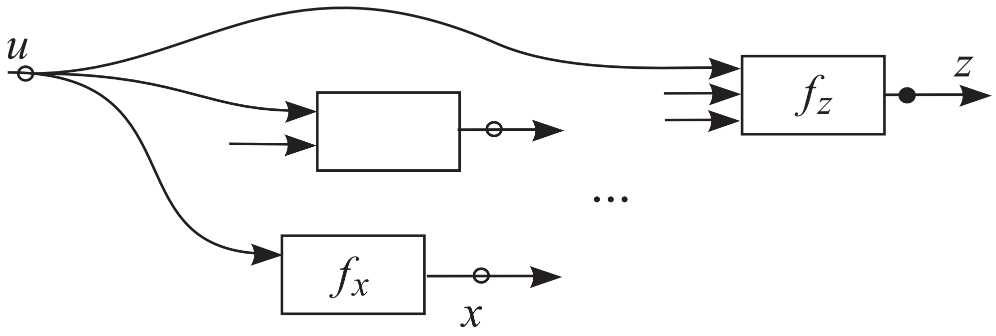


Illustration: ordered systems

$$Q = x_3 = f_3(x_0, x_1, x_2)$$
$$x_2 = f_2(x_0, x_1)$$
$$x_1 = f_1(x_0)$$

FP	BP
$x'_{3\|0} = f'_{3\|0} + f'_{3\|1}\, x'_{1\|0} + f'_{3\|2}\, x'_{2\|0}$	$x'_{3\|0} = f'_{3\|0} + x'_{3\|2}\, f'_{2\|0} + x'_{3\|1}\, f'_{1\|0}$
$x'_{2\|0} = f'_{2\|0} + f'_{2\|1}\, x'_{1\|0}$	$x'_{3\|1} = f'_{3\|1} + x'_{3\|2}\, f'_{2\|1}$
$x'_{1\|0} = f'_{1\|0}$	$x'_{3\|2} = f'_{3\|2}$

for both algorithms

$$x'_{3|0} = f'_{3|0} + f'_{3|1}\, f'_{1|0} + f'_{3|2}\, f'_{2|0} + f'_{3|2}\, f'_{2|1}\, f'_{1|0}$$

Matrix form

$$\mathbf{F}' = \{f'_{i|j}\}_{\substack{i=0,\ldots,n\\ j=0,\ldots,n}} = \begin{bmatrix} 0 & & & & & \\ f'_{1|0} & 0 & & & & \\ f'_{2|0} & f'_{2|1} & 0 & & & \\ \vdots & \vdots & & \ddots & & \\ f'_{n|0} & f'_{n|1} & f'_{n|2} & \cdots & f'_{n|n-1} & 0 \end{bmatrix}$$

forward propagation $\quad (\mathbf{F}' - I)\,\mathbf{X}' = -I$

backpropagation $\quad \mathbf{X}'\,(\mathbf{F}' - I) = -I$

$$\mathbf{X}' = \{x'_{i|j}\}_{\substack{j=0,\ldots,n\\ i=0,\ldots,n}} = \begin{bmatrix} 1 & & & & & \\ x'_{1|0} & 1 & & & & \\ x'_{2|0} & x'_{2|1} & 1 & & & \\ \vdots & \vdots & & \ddots & & \\ x'_{n|0} & x'_{n|1} & x'_{n|2} & \cdots & x'_{n|n-1} & 1 \end{bmatrix}$$

Gradients for two-layer network: influence graph

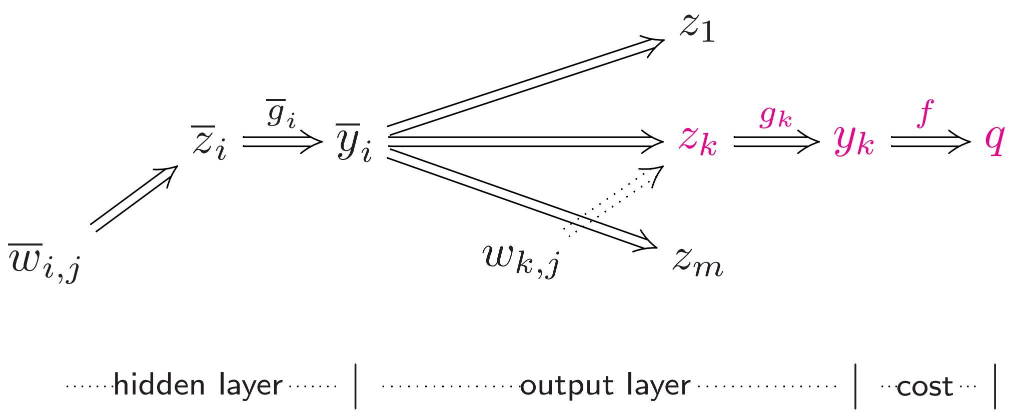
- layer equations

hidden layer $\qquad \overline{y}_i = \overline{g}_i(\overline{z}_i), \quad \overline{z}_i = \sum_{j=0}^{n} \overline{w}_{i,j}\, u_j, \quad i = 1, \ldots, \overline{m}$

output layer $\qquad y_k = g_k(z_k), \quad z_k = \sum_{j=0}^{\overline{m}} w_{k,j}\, \overline{y}_j, \quad k = 1, \ldots, m$

momentary cost $\qquad q = f(y_1, \ldots, y_m)$

- influence graph for $\overline{w}$ i w



- graphs for $\overline{w}$ and w have a common part

Gradients for two-layer network: output layer

$$w_{k,j} \Longrightarrow z_k \xRightarrow{g_k} y_k \xRightarrow{f} q$$

$$\left.\begin{aligned} \frac{dq}{dw_{k,j}} &= \frac{dq}{dz_k}\,\overline{y}_j \\ \frac{dq}{dz_k} &= \frac{dq}{dy_k}\,{g_k}'(z_k) \end{aligned}\right\}\ \text{BP from layer's output to its weights}$$

$$\left.\frac{dq}{dy_k} = f'_k(\mathbf{y})\ \right\}\ \text{cost}$$

$$r_k = f'_k(\mathbf{y})\,{g_k}'(z_k), \qquad \frac{dq}{dw_{k,j}} = r_k\,\overline{y}_j$$

Gradients for two-layer network: hidden layer

$$
\begin{tikzcd}[column sep=small]
 & & & z_1 & & \\
\overline{w}_{i,j} \arrow[r, Rightarrow] & \overline{z}_i \arrow[r, Rightarrow, "\overline{g}_i"] & \overline{y}_i \arrow[ur, Rightarrow] \arrow[r, Rightarrow] \arrow[dr, Rightarrow] & z_k \arrow[r, Rightarrow, "g_k"] & y_k \arrow[r, Rightarrow, "f"] & q \\
 & & & z_m & &
\end{tikzcd}
$$

········hidden layer········ | ········output layer········ | ···cost··· |

$$
\left.
\begin{aligned}
\frac{dq}{d\overline{w}_{i,j}} &= \frac{dq}{d\overline{z}_i}\, u_j \\
\frac{dq}{d\overline{z}_i} &= \frac{dq}{d\overline{y}_i}\, \overline{g}_i{}'(\overline{z}_i)
\end{aligned}
\right\}
\text{ BP inside a layer to its weights}
$$

$$
\left.
\begin{aligned}
\frac{dq}{d\overline{y}_i} &= \textstyle\sum_{k=1}^{m} \frac{dq}{dz_k}\, w_{k,i} \\
\frac{dq}{dz_k} &= \frac{dq}{dy_k}\, g_k{}'(z_k)
\end{aligned}
\right\}
\text{ BP through a layer}
$$

$$
\left.
\frac{dq}{dy_k} = \frac{\partial f(\mathbf{y})}{\partial y_k} = f_k'(\mathbf{y})
\right\}
\text{ cost index}
$$

$$
r_k = f_k'(\mathbf{y})\, g_k{}'(z_k), \qquad \overline{r}_i = \sum_{k=1}^{m} r_k\, w_{k,i}\, \overline{g}_i{}'(\overline{z}_i), \qquad \frac{dq}{d\overline{w}_{i,j}} = \overline{r}_i\, u_j
$$

Gradient BP for two-layer network

(output layer)

$$r_i = f_i'(\mathbf{y})\, {g_i}'(z_i)$$

$$\frac{dq}{dw_{i,j}} = r_i\, \overline{y}_j$$

(hidden layer)

$$\overline{r}_i = \sum_{k=1}^{m} r_k\, w_{k,i}\, {\overline{g}_i}'(\overline{z}_i)$$

$$\frac{dq}{d\overline{w}_{i,j}} = \overline{r}_i\, u_j$$

(output layer)

$$\mathbf{r} = f'(\mathbf{y}) \odot \mathbf{g}'(\mathbf{z})$$

$$\frac{dq}{d\mathbf{W}} = \mathbf{r}\, \overline{\mathbf{y}}^T$$

(hidden layer)

$$\overline{\mathbf{r}} = (\mathbf{W}^T \mathbf{r}) \odot \overline{\mathbf{g}}'(\overline{\mathbf{z}})$$

$$\frac{dq}{d\overline{\mathbf{W}}} = \overline{\mathbf{r}}\, \mathbf{u}^T$$

Example: Gradient BP for standard two-layer network

linear output layer, quadratic momentary cost $q = \frac{1}{2}\|\mathbf{y} - \mathbf{y}^o\|^2$,
activation function $\overline{g}_i(\overline{z}) = 1/(1 + \exp(-\alpha \overline{z})$, $[\overline{g}_i'(\overline{z}) = \alpha\, \overline{y}(1 - \overline{y})]$

(output layer)

$$r_i = y_i - y_i^o$$

$$\frac{dq}{dw_{i,j}} = r_i\, \overline{y}_j$$

(hidden layer)

$$\overline{r}_i = \alpha \sum_{k=1}^{m} r_k\, w_{k,i}\, \overline{y}_i\, (1 - \overline{y}_i)$$

$$\frac{dq}{d\overline{w}_{i,j}} = \overline{r}_i\, u_j$$

(output layer)

$$\mathbf{r} = \mathbf{y} - \mathbf{y}^o$$

$$\frac{dq}{d\mathbf{W}} = \mathbf{r}\, \overline{\mathbf{y}}^T$$

(hidden layer)

$$\overline{\mathbf{r}} = \alpha\, (\mathbf{W}^T \mathbf{r}) \odot \overline{\mathbf{y}} \odot (1 - \overline{\mathbf{y}})$$

$$\frac{dq}{d\overline{\mathbf{W}}} = \overline{\mathbf{r}}\, \mathbf{u}^T$$

Gradients in multi-layer network

layer ℓ, $\ell = 1, \dots, L$ $\quad z_i^\ell = \sum_{j=0}^{n^\ell} w_{i,j}^\ell \, y_j^{\ell-1}, \quad y_i^\ell = g_i^\ell(z_i^\ell), \quad i = 1, \dots, m^\ell$

cost $\quad q = f(y_1^L, \dots, y_m^L)$

$$w_{i,j}^{\ell-1} \Rightarrow z_i^{\ell-1} \xRightarrow{g_i^{\ell-1}} y_i^{\ell-1} \Rightarrow \{z_1^\ell, \; z_k^\ell, \; z_{m^\ell}^\ell\}, \quad z_k^\ell \xRightarrow{g_k^\ell} y_k^\ell \cdots\cdots y_p^{L-1} \Rightarrow \{z_1^L, \; z_p^L, \; z_{m^L}^L\}, \quad z_p^L \xRightarrow{g_p^L} y_p^L \xRightarrow{f} q$$

layer $\ell - 1$ | layer ℓ | .... | output layer L | cost |

$$\left.\begin{aligned} \frac{dq}{dw_{i,j}^{\ell-1}} &= \frac{dq}{dz_i^{\ell-1}} u_j^{\ell-1} \\ \frac{dq}{dz_i^{\ell-1}} &= \frac{dq}{dy_i^{\ell-1}} g_i^{\ell-1\prime}(z_i^{\ell-1}) \end{aligned}\right\} \text{BP in a layer } \ell-1 \text{ to its weights}$$

$$\left.\begin{aligned} \frac{dq}{dy_i^{\ell-1}} &= \sum_{k=1}^{m^\ell} \frac{dq}{dz_k^\ell} w_{k,i}^\ell \\ \frac{dq}{dz_k^\ell} &= \frac{dq}{dy_k^\ell} g_k^{\ell\prime}(z_k^\ell) \end{aligned}\right\} \text{BP through a layer } \ell$$

$$\left.\frac{dq}{dy_p^L} = \frac{\partial f(\mathbf{y})}{\partial y_p^L} = f_p'(\mathbf{y})\right\} \text{cost index}$$

Gradient BP in multi-layer network

cost index

$$\frac{dq}{dy_p} = \frac{\partial f(\mathbf{y})}{\partial y_p} = f_p'(\mathbf{y})$$

$$\frac{dq}{d\mathbf{y}} = \frac{\partial f(\mathbf{y})}{\partial \mathbf{y}} = f'(\mathbf{y})$$

BP through the layer

$\ell = L, \ldots, p+1$

$$\frac{dq}{dy_i^{\ell-1}} = \sum_{k=1}^{m^\ell} \frac{dq}{dy_k^\ell}\, g_k^{\ell\,\prime}(z_k^\ell)\, w_{k,i}^\ell \qquad \ell = L, \ldots, p+1$$

$$\frac{dq}{d\mathbf{y}^{\ell-1}} = \mathbf{W}^{\ell\,T}\left(\frac{dq}{d\mathbf{y}^\ell} \odot \mathbf{g}^{\ell\,\prime}(\mathbf{z}^\ell)\right) \qquad \ell = L, \ldots, p+1$$

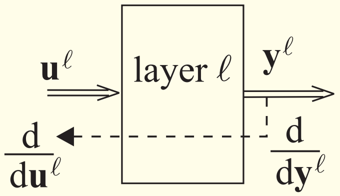
BP in a layer p to its weights

$$\frac{dq}{dw_{i,j}^p} = \frac{dq}{dy_i^p}\, g_i^{p\,\prime}(z_i^p)\, u_j^p$$

$$\frac{dq}{d\mathbf{W}^p} = \left(\frac{dq}{d\mathbf{y}^p} \odot \mathbf{g}^{p\,\prime}(\mathbf{z}^p)\right) \mathbf{u}^{p\,T}$$

BP operator

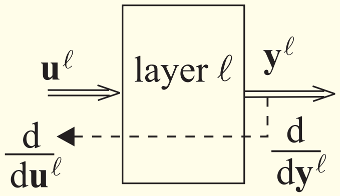
BP to layer's weights



$$\nabla_{\mathbf{W}} = \mathcal{B}_{\mathbf{W}}(\nabla_{\mathbf{y}}) = \Big(\nabla_{\mathbf{y}} \odot \mathbf{g}'(\mathbf{z})\Big)\, \mathbf{u}^T$$

$$\nabla_{\mathbf{W}} = \frac{d\,\cdot}{d\mathbf{W}} \qquad \nabla_{\mathbf{y}} = \frac{d\,\cdot}{d\mathbf{y}} \qquad \nabla_{\mathbf{u}} = \frac{d\,\cdot}{d\mathbf{u}}$$

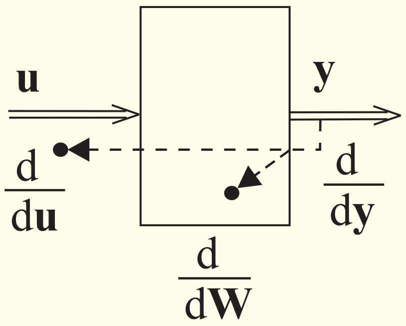
BP through the layer



$$\nabla_{\mathbf{u}} = \mathcal{B}_{\mathbf{u}}(\nabla_{\mathbf{y}}) = \mathbf{W}^T\Big(\nabla_{\mathbf{y}} \odot \mathbf{g}'(\mathbf{z})\Big)$$

BP module for a layer

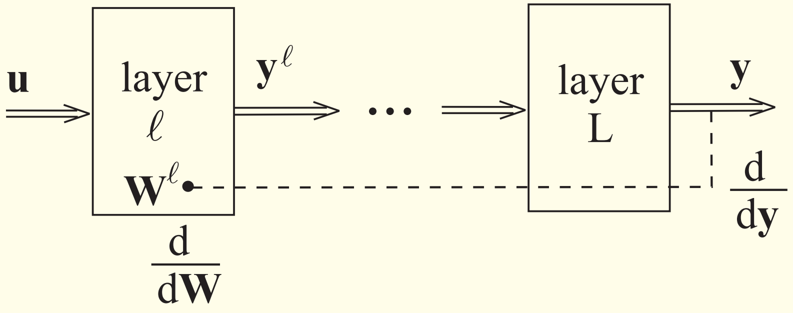
$$(\nabla_{\mathbf{W}}, \nabla_{\mathbf{u}}) = \mathcal{B}(\nabla_{\mathbf{y}})$$



$$\mathbf{r} = \nabla_{\mathbf{y}} \odot \mathbf{g}'(\mathbf{z})$$
$$\nabla_{\mathbf{W}} = \mathbf{r}\,\mathbf{u}^T$$
$$\nabla_{\mathbf{u}} = \mathbf{W}^T\mathbf{r}$$

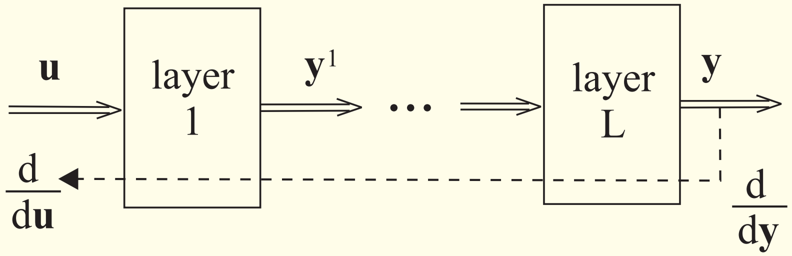
BP through the network

BP through the network to the weights



$$\nabla_{\mathbf{W}^\ell} = \mathcal{B}_{\mathbf{W}^\ell}\,\mathcal{B}_{\mathbf{u}^{\ell+1}} \ldots \mathcal{B}_{\mathbf{u}^L} \nabla_{\mathbf{y}^L}$$

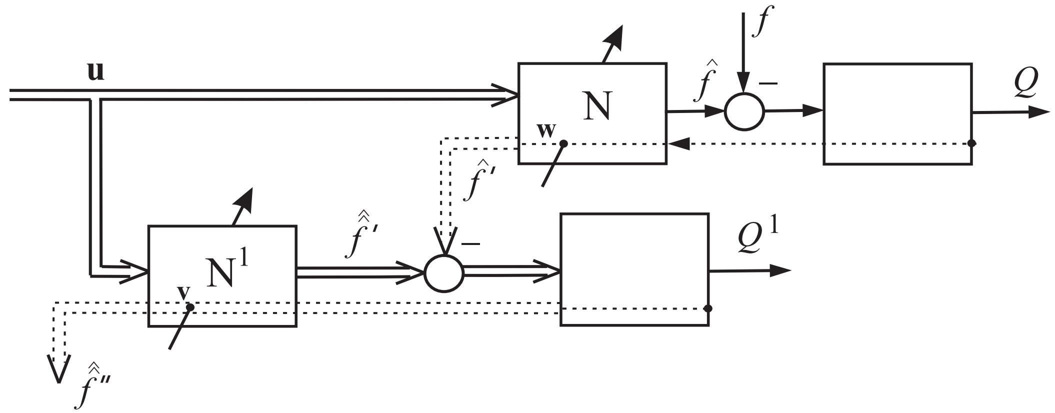
BP through the network



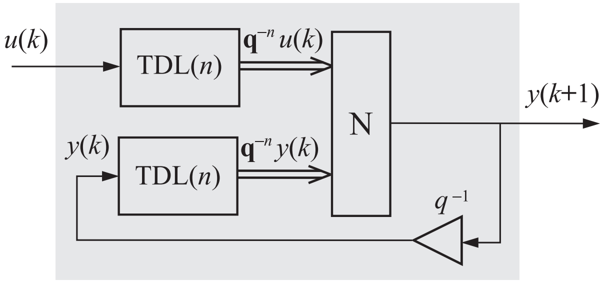
$$\nabla_{\mathbf{u}^1} = \mathcal{B}_{\mathbf{u}^1}\,\mathcal{B}_{\mathbf{u}^2} \ldots \mathcal{B}_{\mathbf{u}^L} \nabla_{\mathbf{y}^L}$$

Hessian

- $y^o = f(\mathbf{u})$ is desired for $\mathbf{N}$, input $\mathbf{u}$
- $\mathbf{N}$ approximates f: $\mathbf{N}(\mathbf{u}) = \widehat{f}(\mathbf{u})$
- BP through the network $\mathbf{N}$ calculates values of the gradient of function $\widehat{f}$:
  $\widehat{f}'(\mathbf{u}) = \dfrac{d\mathbf{N}(\mathbf{u})}{d\mathbf{u}} \approx f'(\mathbf{u})$
- $\widehat{f}'(\mathbf{u})$ are desired for the network $\mathbf{N}^1$, input $\mathbf{u}$
- $\mathbf{N}^1$ approximates $\widehat{f}'$: $\mathbf{N}^1(\mathbf{u}) = \widehat{\widehat{f}'}$
- BP through the network $\mathbf{N}^1$ approximates values of approximated Hessian $\widehat{\widehat{f}''}$



Gradient backpropagation in NARX network



- SISO NARX network

$$y(t+1) = \mathbf{N}\big(y(t), \ldots, y(t-n), u(t), \ldots, u(t-n);\ \mathbf{w}\big)$$

- cost $Q = \sum_{t=1}^{N} f(y(t))$

- w influences network's output at all moments t

$$\frac{dQ}{dw} = \sum_{t=1}^{N} \frac{dQ}{dy(t)} \frac{\partial \mathbf{N}(t)}{\partial w}$$

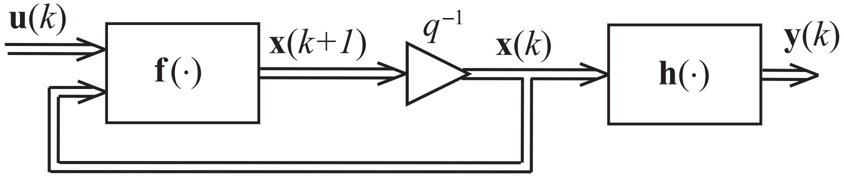
- $y(t)$ wpywa na $y(t+1), \ldots, y(t+n)$ (through TDL and the network) and cost

$$\frac{dQ}{dy(t)} = \sum_{s=t+1}^{t+n} \frac{dQ}{dy(s)} \frac{\partial \mathbf{N}(s)}{\partial y(t)} + f'(t)$$

- calculation $\dfrac{dQ}{dy(t)}$ *in the reversed time* for $t = N, \ldots, 1$, boundary conditions $\dfrac{dQ}{dy(s)} = 0$ for $s = N+1, \ldots, N+n$

- calculation of $\dfrac{\partial \widehat{\psi}(t)}{\partial w}$: BP through the network

Networks in a dynamic system



- dynamic system - state and output equations

$$\mathbf{x}(t+1) = \mathbf{N}^{\mathbf{f}}\big(\mathbf{x}(t), \mathbf{u}(t);\ \mathbf{w}\big)$$
$$\mathbf{y}(t) = \mathbf{N}^{\mathbf{h}}\big(\mathbf{x}(t);\ \mathbf{v}\big)$$

- cost $Q = \frac{1}{2}\sum_{t=1}^{N} \|\mathbf{y}(t) - \mathbf{y}^{o}(t)\|^2$

Gradient calculation by forward propagation

- Q directly depends on output coordinates at all times

$$\frac{dQ}{dw} = \sum_{t=1}^{N} \sum_{j=1}^{n} \frac{\partial Q}{\partial y_j(t)} \frac{dy_j(t)}{dw}$$

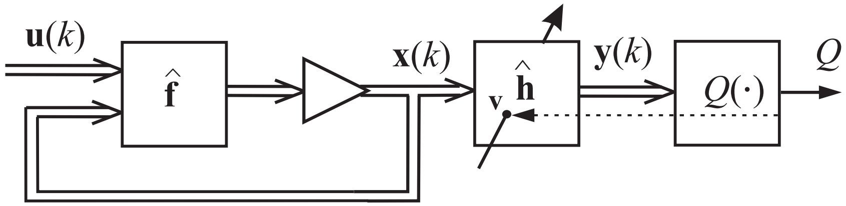
- every input depends on all state coordinates *at the same moment*

$$\frac{dy_i(t)}{dw} = \sum_{j=1}^{p} \frac{\partial \widehat{h}_i(t)}{\partial x_j(t)} \frac{dx_j(t)}{dw}$$

- state depends on weights, directly and indirectly through all previous state values

$$\frac{dx_i(t)}{dw} = \frac{\partial f_i(t)}{\partial w} + \sum_{j=1}^{n} \frac{\partial \widehat{f}_i(t)}{\partial x_j(t-1)} \frac{dx_j(t-1)}{dw}$$

Gradient backpropagation – output network



- v influences all output coordinates at all times t

$$\frac{dQ}{dv} = \sum_{t=1}^{N} \sum_{j=1}^{p} \frac{dQ}{dy_j(t)} \frac{\partial \mathbf{N}_j^h(t)}{\partial v}$$

- every coordinate of $y(t)$ directly influences cost

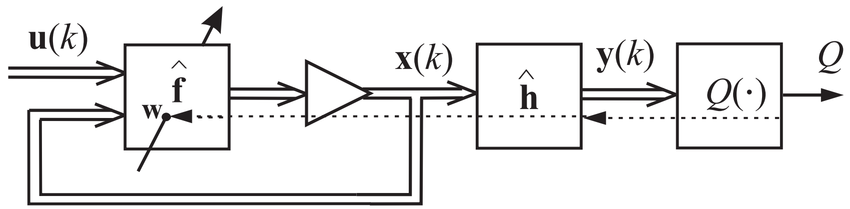
$$\frac{dQ}{dy_i(t)} = y_i(t) - y_i^o(t)$$

- finally

$$\frac{dQ}{dv} = \sum_{t=1}^{N} \sum_{j=1}^{n} \left(y_i(t) - y_i^o(t)\right) \frac{\partial \mathbf{N}_j^h(t)}{\partial v}$$

- BP can be also used to calculate derivatives $\dfrac{\partial \mathbf{N}_j^h(t)}{\partial v}$

Gradient backpropagation – state network



- w influences all state coordinates at all times t

$$\frac{dQ}{dw} = \sum_{t=1}^{N} \sum_{j=1}^{n} \frac{dQ}{dx_j(t)} \frac{\partial \mathbf{N}_j^f(t)}{\partial w}$$

- $x_i(t)$ influences $\mathbf{y}(t)$ and (except at $t = N$) $\mathbf{x}(t+1)$

$$\frac{dQ}{dx_i(t)} = \sum_{j=1}^{p} \frac{dQ}{dy_j(t)} \frac{\partial \mathbf{N}_j^h(t)}{\partial x_i(t)} + \sum_{j=1}^{n} \frac{dQ}{dx_j(t+1)} \frac{\partial \mathbf{N}_j^f(t)}{\partial x_i(t)} \qquad \text{dla } t < N$$

$$\frac{dQ}{dx_i(N)} = \sum_{j=1}^{p} \frac{dQ}{dy_j(N)} \frac{\partial \mathbf{N}_j^h(N)}{\partial x_i(N)}$$

- $y(t)$ directly influences momentary cost

$$\frac{dQ}{dy_i(t)} = y_i(t) - y_i^o(t)$$

Gradients backpropagation in dynamic systems

- sensitivity vectors

$$Q_{\mathbf{x}}(t) = \left[\frac{dQ}{dx_1(t)} \quad \cdots \quad \frac{dQ}{dx_n(t)} \right]^T$$

- linear equations in the *reversed time*

$$Q_{\mathbf{x}}(t) = \mathbf{N}_{\mathbf{x}}^{\mathbf{f}}(t)^T Q_{\mathbf{x}}(t+1) + \mathbf{N}_{\mathbf{x}}^{\mathbf{h}}(t)^T \big(\mathbf{y}(t) - \mathbf{y}^o(t)\big), \qquad Q_{\mathbf{x}}(N+1) = \mathbf{0}$$

$$Q_{\mathbf{w}}(t) = Q_{\mathbf{w}}(t+1) + \mathbf{N}_{\mathbf{w}}^{\mathbf{f}}(t)^T Q_{\mathbf{x}}(t), \qquad Q_{\mathbf{w}}(N+1) = \mathbf{0}$$

$$Q_{\mathbf{v}}(t) = Q_{\mathbf{v}}(t+1) + \mathbf{N}_{\mathbf{v}}^{\mathbf{h}}(t)^T \big(\mathbf{y}(t) - \mathbf{y}^o(t)\big), \qquad Q_{\mathbf{v}}(N+1) = \mathbf{0}$$

where $\dfrac{dQ}{dw}$ i $\dfrac{dQ}{dv}$ are elements of $Q_{\mathbf{w}}(1)$, $Q_{\mathbf{v}}(1)$