Adaptive modeling of biochemical pathways

R.Brause
J.W.G.University, Frankfurt, Germany
Brause@cs.uni-frankfurt.de

Abstract

In bioinformatics, biochemical pathways can be modeled by many differential equations. It is still an open problem how to fit the huge amount of parameters of the equations to the available data. Here, the approach of systematically learning the parameters is necessary.

In this paper, for the small, important example of inflammation modeling a network is constructed and different learning algorithms are proposed. It turned out that due to the nonlinear dynamics evolutionary approaches are necessary to fit the parameters for sparse, given data.

1. Introduction

In living organisms many metabolisms and immune reactions depend on the biochemical pathways of molecules. In Bioinformatics, these pathways are modeled by many differential equations. For complicated systems, differential equations systems (DES) with up to 7,000 equations and 20,000 associated parameters exist and model reality. The motivation for life science industry to use such systems is evident: A prediction of reactions and influences by simulated models helps avoiding time-consuming, expensive animal and laboratory experiments, decrease the high costs for developing new drugs and therefore may save millions of Euros.

Although the basic idea is quite seducing, the practical problems associated with the simulation approach are difficult to solve: How can all parameters be set to the correct values? And if all parameters are different for each individual, how can they be adapted to the real values based only on a small set of measured data per organism? Here, neural networks come into hand.

Traditionally, the worlds of neural networks and of differential equations are well separated. Neural networks are used to approximate stochastically unknown functions and relations. This approximation is a kind of implicit model of the unknown dependencies. In contrast, differential equations are used to model explicitly all relations. Here, the given interdependencies will result in a complex, parameter determined behavior.

Now, what about using the approximation capabilities of neural networks for approximating the unknown parameters? This idea is elaborated in the next section.

2. The differential equation neural network of inflammation and septic shock

The initial idea of using neural networks for approximating the parameters of differential equations can be reformulated in an interesting manner. It is well known that all activity equations and learning equations of neural networks can be formulated as difference equations. Now, if we can reformulate our differential equation system DES by difference equations and map the difference equations on neural networks, perhaps we can learn the parameters automatically by the emulating neural networks instead of setting them manually in a long and laborious way.

Let us take a more concrete look on an example system: the DES of inflammation and septic shock. The symptoms of septic shock contain low blood pressure, high ventilation and high heart rates and may occur after an infection or a trauma (damage of tissue). The septic shock research has no convincing results yet; there is still a high mortality of about 50% on the intensive care units (ICU) and nobody knows why. It is only possible to predict the outcome for a patient in advance just for 3 days, see [3] . In 1999, about 250,000 deaths were associated with sepsis in the USA.

A confusing myriad of chemical pathways and molecules of the immune system produce the septic shock state. A recent approach tries to model one of the main pathways by a small set of 16 ordinary differential equations [1]. Nevertheless, this model uses 117 parameter constants that are far too many for a simple approach. Therefore, for studying some approximation methods we restrict ourselves first to a simplified but still functional version of the model that uses only three variables and 12 constant parameters [2]. First, let us introduce the basic model assumptions.

Let P be the pathogen influence and M the immunological response, e.g. the macrophages involved. Then

1) P will be increased by cell growth, i.e. splitting of pathogen cells: $P'(t) \sim P$
2) For this, there is a limit of resources, i.e. $P'(t) \sim (P_{max}-P)$ with concentration $P_{max} = 1$.
3) P will also decrease by the probability that macrophages and pathogens are at the same place: $P'(t) \sim -M \cdot P$

4) The number of macrophages will grow when a "combat indicator" is produced when they destroy the pathogen. Therefore, they grow with the probability of macrophages and pathogens at the same place: $M'(t) \sim M \cdot P$
5) Macrophages die at constant rate: $M'(t) \sim -M$
6) There is a cell damage D which is caused by inflammation. This will also cause the number of macrophages to grow because they are also used to do a "cleaning service". Like the pathogens, the macrophages grow with the probability of being at the same place: $M'(t) \sim MD$
7) There is a limited resource for macrophages. Therefore, all growth is also proportional to the remaining resource: $M'(t) \sim (1-M)$
8) The cell damage is repaired with a certain rate: $D'(t) \sim -D$
9) Let us assume that the amount of additional damage is indicated by a sigmoid function h() of the number of macrophages: $D'(t) \sim h(M-\theta)$ where θ is a threshold.

All the assumptions can be combined into a coupled system of three first order differential equations:

$$P'(t) = \alpha_1 P(1-P) - \alpha_2 MP \qquad 1)+2)+3) \qquad \alpha_i>0 \quad (1)$$
$$M'(t) = -\beta_1 M + M(1-M)(\beta_2 P + \beta_3 D) \qquad 4)+5)+6)+7) \qquad \beta_i>0 \quad (2)$$
$$D'(t) = -\chi_1 D + \chi_2 h((M-\theta)/\chi_3) \qquad 8)+9) \qquad \chi_i>0 \quad (3)$$

where a typical parameter regime takes as maximal values

α_1: 0.1	β_1: 1.0	χ_1: 0.1	$h(.)= 1/(1+\exp(.))$
α_2: 1.0	β_2: 10.0	χ_2: 0.04	$\theta = 0.5$
	β_3: 1.0	χ_3 : 0.25	

For discrete time dynamics, the differential equations have to be transformed into difference equations. In general, an equation of the form

$$\frac{dy_i}{dt} = f_i(y_1,\ldots,y_n,t) \qquad i=1,..n$$

can be written by a difference equation

$$y_i(t+h) = y_i(t) + h\, f_i(y_1,\ldots,y_n,t) + O(h^2)$$

For small values of *h*, the last term in this Taylor expansion becomes very small and can be neglected (*finite difference method*). Therefore, for discrete, relative small time steps t, t+1, t+2, … we might replace the differential equations by the difference equations

$$P(t+1) = P + a_1(1-P)P + a_2 MP \quad \text{with } a_1:=\alpha_1,\ a_2:=-\alpha_2 \quad (4)$$
$$M(t+1) = a_3 M + a_4 M(1-M)P + a_5 M(1-M)D$$
$$\text{with } a_3:=(1-\beta_1),\ a_4:=\beta_2,\ a_5:=\beta_3 \quad (5)$$

$$D(t+1) = a_6 D + a_7\, h(a_8(M-a_9))$$
$$\text{with } a_6:=(1-\chi_1),\ a_7:=\chi_2,\ a_8:=\chi_3^{-1},\ a_9:=\theta \quad (6)$$

This can also be interpreted as the activity of a recurrent two-layer neural network with the structure drawn in Fig.1. The sum of each equation is emulated by one linear neuron (Σ-neuron) in the second layer with the weighted input of non-linear terms. Each non-linear term is produced by one non-linear neuron of a previous layer.

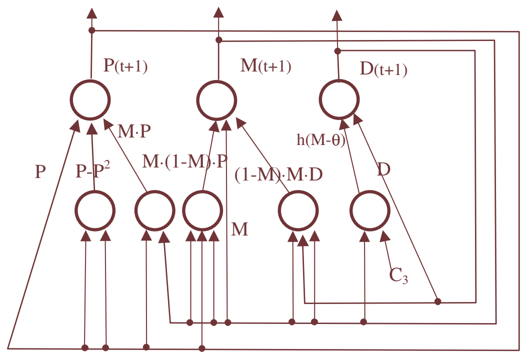


Fig. 1 A recurrent neural network directly modeling the system

There is another version of the differential equations, which can be written in polynomial form:

$$P(t+1) = w_1 P + w_2 MP + w_3 P^2$$
$$\text{with } w_1:=(1+\alpha_1),\ w_2:=-\alpha_2,\ w_3:=-\alpha_1 \quad (7)$$

$$M(t+1) = w_4 M + w_5 MP + w_6 MD + w_7 M^2 P + w_8 M^2 D$$
$$\text{with } w_4:=(1-\beta_1),\ w_5:=\beta_2,\ w_6:=\beta_3,\ w_7:= -\beta_2,\ w_8:= -\beta_3 \quad (8)$$
$$D(t+1) = w_9 D + w_{10}\, h(w_{11}(M-w_{12}))$$
$$\text{with } w_9:=(1-\chi_1),\ w_{10}:=\chi_2,\ w_{11}:= \chi_3^{-1},\ w_{12}:=\theta \quad (9)$$

The corresponding network is shown in Fig. 2. Here, the first layer is mainly composed of Π-neurons which compute the product of its input terms.

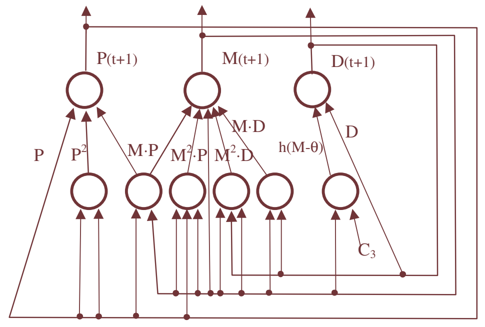


Fig. 2 The polynomial recurrent neural network modeling the system

The plot of the time course for the three outputs (three variables) for the set of parameters shown in Tab. 1 is shown in Fig. 3.

Tab. 1 The constant parameter values

$\alpha_1 = 0.045$	$\beta_1 = 1.0$	$\chi_1 = 0.087$
$\alpha_2 = -0.225$	$\beta_2 = 6.0$	$\chi_2 = 0.04$
	$\beta_3 = 0.95$	$\chi_3 = 0.25$

It can be concluded that an infection (P) causes cell damage (D) and a delayed activity of the macrophages (M). The infection is defeated by the macrophages which decrease to a sufficient level afterwards. In this case (parameter regime), the infection remains chronically.

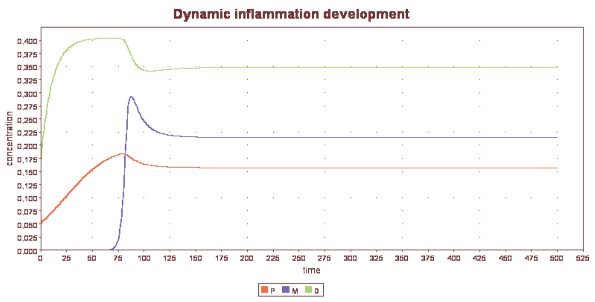


Fig. 3 The time dynamics of the equations (4),(5) and (6).

Now, how can the parameters, which correspond to the weights of the second layer, be learned? It is well known that the non-linear transfer function of deterministic chaotic systems can be efficiently learned in order to predict a chaotic time series, see for instance [5]. Therefore, all dynamics which evolve by recurrent influence may be modeled by recurrent neural nets containing delayed signals, implemented e.g. in the discrete case by delay elements like tapped delay lines. In this case, the learning can be done by simple error reducing algorithms.

In the next section we regard the adaptation of the parameters more closely.

3. Learning the parameters

Generally, the biochemical pathways are very complex. It is not clear, which influences are important and which are not important. For the analytical description of equations 1 and 2 this means that the number of terms and the values of its parameters are not given *a priori*, but have to be estimated ("learned") by the real observed data. How can this be done?

The learning process has to take into account that the error of the approximation is caused either by the value of the parameter or the existence of the term. Since decreasing a very small term (very small parameter) is equivalent to dropping it, the goal of selecting the most important terms is also reached by minimizing the error influence of each term adjusting its parameter value.

To describe the possible learning algorithm in detail, let us first generalize the three equations (4),(5),(6) to

$$y_1(t+1) = a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15}x_5 + a_{16}x_6 + a_{17}x_7 + a_{18}x_8 \qquad (10)$$

$$y_2(t+1) = a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 + a_{25}x_5 + a_{26}x_6 + a_{27}x_7 + a_{28}x_8 \qquad (11)$$

$$y_3(t+1) = a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 + a_{35}x_5 + a_{36}x_6 + a_{37}x_7 + a_{38}x_8 \qquad (12)$$

with

$x_1 := P$, $x_2 := (1-P)P$, $x_3 := MP$, $x_4 := M$, $x_5 := M(1-M)P$, $x_6 := M(1-M)D$, $x_7 := D$, $x_8 := h(M-\theta)$

and

$a_{11}=1$, $a_{12}=a_1$, $a_{13}=a_2$, $a_{14}=a_{15}=a_{16}=a_{15}=a_{17}=a_{18}=0$

$a_{21}=a_{22}=a_{23}=0$, $a_{24}=a_3=1-\beta_1$, $a_{25}=a_4=\beta_2$, $a_{26}=a_5=\beta_3$, $a_{27}=a_{28}=0{,}3$

$a_{31}=a_{32}=a_{33}=a_{34}=a_{35}=a_{36}=0$, $a_{37}=a_6=1-\chi_1$, $a_{38}=a_7=\chi_2$

In conclusion, the system of parameters to be adapted can be described by the linear equation

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

$$\text{with } \mathbf{y} := (y_1,\ldots,y_m),\ \mathbf{x} := (x_1,\ldots,x_n),\ \mathbf{W}=(a_{ij}) \qquad (13)$$

Let us assume that we have for t=1,…,N data samples the input P(t),M(t),D(t) i.e. **x**, and the desired output **L**. As learning goal, the weights **W** should be adapted such that for **x**(t), **L**(t) the mean squared error of the predicted output to all observed samples at time points t

$$R(\mathbf{W}) := \langle(\mathbf{y}(\mathbf{x}(t)) - \mathbf{L}(t))^2\rangle_t = \langle(\mathbf{W}\mathbf{x}(t)) - \mathbf{L}(t))^2\rangle_t = \min$$

takes a minimum.

Now, the straightforward approach as performed in [5] trains the net with several hundred data tuples (input, desired output) and learns the nonlinear mapping from the input to the output for each time step, i.e. each data tuple. There are several possible learning algorithms to solve this task.

The simple gradient learning takes the gradient of R to compute the next iteration step k

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \gamma(k)\ \text{grad}_w\, R(\mathbf{W})$$

For the mean squared error this becomes with the expectation brackets $\langle.\rangle$

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \gamma(k)\ \langle 2(\mathbf{W}\mathbf{x}(t)) - \mathbf{L}(t))\mathbf{x}(,t)\rangle_t \qquad (14)$$

This equation gives only a crude direction in which the update should evolve. It can be smoothed in order to reflect the proper error amount, see [4], to

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \gamma(k) \left\langle (\mathbf{W}\mathbf{x}(t)) - \mathbf{L}(t)) \frac{\mathbf{x}(t)}{|\mathbf{x}(t)|^2} \right\rangle_t$$

Widrow-Hoff learning (15)

For the weights a_{ij} of eq.(10)-(12) this becomes for the k-th time step the stochastic approximation

$$a_{ij}(k+1) = a_{ij}(k) + \gamma(L_i - y_i(t))x_j(t)/\Sigma_p x_p(t)^2 \qquad (16)$$

Although the input **x** is produced by nonlinear terms (polynomials of P,M,D), the learning is only used for the linear mapping by matrix **W**. Therefore, knowing the desired output {P(t),M(t),D(t)} and by this the set {**x**(t)}, we can directly learn the parameters **W** of the mapping.

Please consider that the iteration is deterministic; for each iteration step, the average over N samples must be computed. For a small number of data, e.g. for N=6 for rat blood samples, this is easy to do. In the case of the many data of a complete time course, the stochastic approximation approach should be chosen and the expectation brackets be dropped as done in eq.(16).

In the deterministic case of a DES with no random influences, the parameters can even be directly computed: *m* samples give us *m* equations for *m* parameters that are easily computable by the well-known Gaussian elimination method.

The gradient learning has already been used successfully in learning the parameters of nonlinear chaotic mappings, see e.g. [5]. Therefore, we will not consider it further in detail. Instead, let us refine our problem in the next section.

4. Learning from sparse data

Let us assume that we do not have the full data set of Fig. 3 but only the small set of observed data given in table 1.

Time step	P	M	D
0	0.050000	0.001000	0.150000
100	0.201215	0.206079	0.254347
200	0.183751	0.206844	0.342027
300	0.177270	0.206750	0.374282
400	0.174876	0.206680	0.386141
500	0.173995	0.206649	0.390500

Tab. 2 The observed sparse data

This situation is different from the previous one of learning the unknown parameters: the time scales of the observed training data and of the iteration cycles are different. For instance, the dynamics of inflammation might be in the reach of hours, whereas the observed data is taken once each day. In Fig. 4 this situation is shown. Here, the variable y(t) changes after each time tick, but it is only measured at time points t_i.

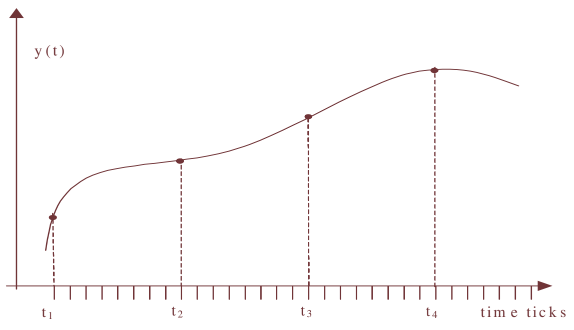


Fig. 4 The different time intervals for the differential equation and the observations

The different time scales will change heavily the approximated coefficients and difference equations. To see this, let us consider an example. For the differential equations

$$\dot{y} = axy\,,\ \dot{x} = bxy$$

we get the difference equations

$$\dot{y} = axy \approx \frac{y(t+\Delta t) - y(t)}{\Delta t} \text{ or } y(t+\Delta t) = y(t) + \Delta t \cdot axy$$

and $$\dot{x} = bxy \approx \frac{x(t+\Delta t) - x(t)}{\Delta t} \text{ or } x(t+\Delta t) = x(t) + \Delta t \cdot bxy$$

For $\Delta t=1$ we get the differential equations $y(t+1) = y(t) + axy$ and $x(t+1) = x(t) + bxy$.
This means

$$y(t+2) = y(t+1) + axy(t+1) = y + axy + a(x + bxy)(y + axy)$$
$$= y + 2axy + a^2x^2y + abxy^2 + a^2bx^2y^2$$

whereas for $\Delta t=2$ we get

$$y(t+2) = y + 2axy$$

which is significantly different from the previous equation.

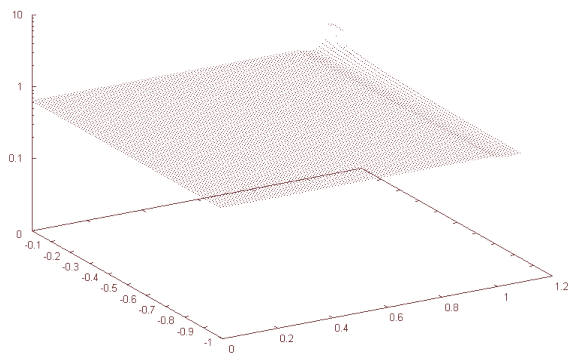
Therefore, if we ignore the time steps between the observations and assume that the network iterates once for one observations we will not be able to predict the best

fitting parameters a_i for the difference equations which have several time steps between the observations.
Now, how can we proceed to approximate the unknown parameters from sparse observations? Obviously, the direct approach of [5] is not possible here because we have no training tuples (**y**(t),**y**(t+1)). Our approach has to be different.

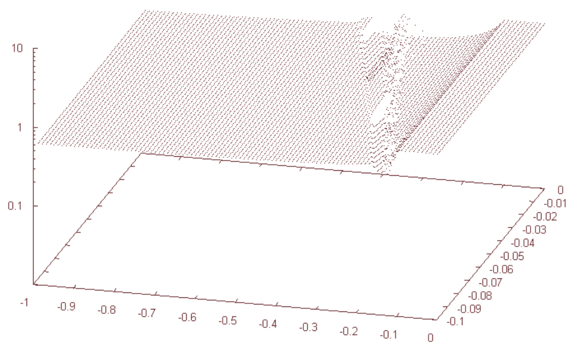
4.1 The error landscape

What can we expect for any error-decreasing learning? What does the error landscape look like for our problem? In Fig. 5 different aspects of the error landscape are shown. The figures are obtained by varying two weights (parameters) in the difference equations within the given interval, the remaining weights are fixed at the boundaries.

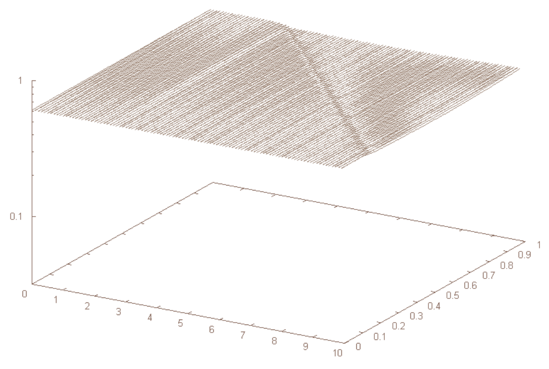
The squared error between the demanded values of P, M, D (see Tab. 2) and the computed ones at t=100 is plotted in logarithmic scaling for the z-direction.



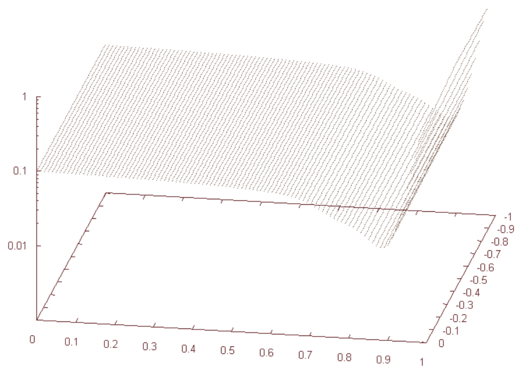
(1) weights w_1 vs. w_2



(2) weights w_2 vs. w_3



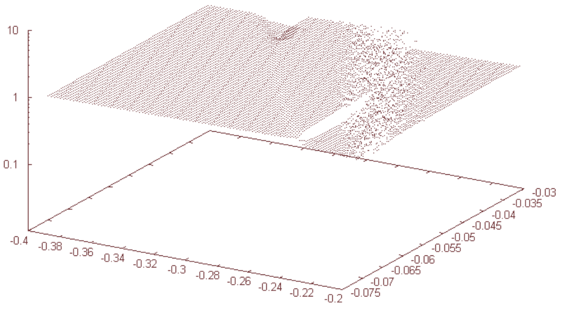
(3) weights w_5 vs. w_6



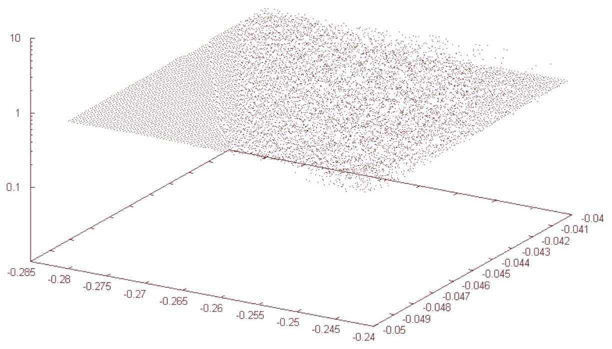
(4) weights w_8 vs. w_9

Fig. 5 Types of the error landscape

It is interesting to see that the error landscape is highly nonregular. There are wide areas where the error does not change; the interesting regions (valleys) of the error are in a very small parameter range, see e.g. at the boarder in Fig. 5 (1) or diametral in Fig. 5 (3). By these statements one might think that simple gradient seeking mechanisms might be sufficient. But additionally, we have a highly dynamic system with fixed upper and lower limits of the dynamical variables. This might produce chaotic behaviour, see [6]. In our case, we encounter chaos indeed. To see this, let us zoom in on the rough region of Fig. 5 (2).



(1) First zoom



(2) Second zoom

Fig. 6 Zooming in a chaotic landscape

A smooth landscape change by a complete nonregular landscape which can not be anticipated by any standard minimum-seeking mechanism.

4.2 Random adaptive learning

For all cases where the analytic form of the objective function is not given and the error landscape is not smooth we can not compute the gradient explicitly. Instead, let us consider a variant of the classical evolutionary approach as it was introduced by Rechenberg 1973 [7].

1. Generate a new set of random weights numerically by incrementing the old value with a random number, e.g. a Gaussian deviation.
2. test it: does it decrease the objective function?
3. If no, inverse the sign of the increment and test it: does it decrease the objective function?
4. If no, take the old weight values and choose another weight set.

The advantage of this approach is its independency of the complexity of the objective function. The disadvantage is its slow convergence speed: we have to recompute the objective function each time we change only one weight, and we can not adapt the step width in advance.

For our parameter approximation problem, this results in the following algorithm:

P-LEARN :
```
for i=1 to m do
   w0[i]:=0.9*(MaxValue[i]-MinValue[i])
              + MinValue[i]
next i
R0:= Risk(k);
for t:=1 to 100,000 do
  for i:=1 to m do
       w[i]:= w0[i]
              + GaussRandom()*s
              *(MaxValue[i]-MinValue[i])
  next i
  R:= Risk(k);
  if R<R0 then
    for i:=1 to m do w0[i]=w[i] next i;
    R0=R;
  endif
next t
```

Fig. 7 The evolutionary learning algorithm in pseudo code notation

In the beginning, the weights (parameters) are initialized to random values within the allowed range. Then, the algorithm uses the Risk(k) method to activate the neural network feedback (iterate the differential equation) and saves the output at the predefined time points, ending with the k-th point. By the difference between the simulation output and the given real world observations the absolute error, the scalar R, is computed and used for changing the weights.

There is another problem to be solved. Since the value of the 100st iteration is a highly nonlinear function of the starting values, the time course can easily grow out of bounds. Therefore, at each iteration we have to assure the upper and lower limits 1.0 and 0.0 for the concentrations.

4.3 Simulating Random Parallel Learning

Now, let us simulate the evolutionary approach for approximating the parameters of the differential equation system of section 2. In Fig. 8 the error decrease is shown. In order to compute this, the weights are initialized to half the maximal range. Then, random deviations of the weights are computed with a normal distribution of variance $\sigma = s \cdot$(weight range). Executing the algorithm of Fig. 7 the resulting squared error is stored for each time step. The procedure is performed 100 times for each s, giving 100 time courses. The average time course of 10,000 steps (mutations) for each value of s is plotted in Fig. 8.

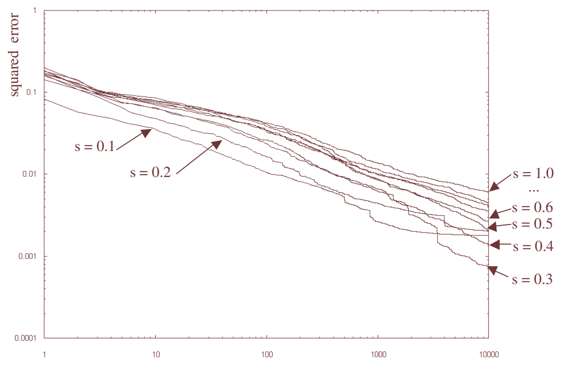


Fig. 8 Average error decrease and random variance

The figure shows that the random variance s above a value of s=0.3 performs well. Nevertheless, the convergence is very slow. Does this accelerate if we mutate only one weight after the other?

4.4 Simulating Random Sequential Learning

Instead of changing all weights for the next try, let us change only one. This might leave some weights in a "good" state, while changing non-optimal ones. Fig. 9 shows a serial version of the parallel evolutionary algorithm of Fig. 7.

S-LEARN :
```
for i=1 to m do
   w0[i]:=0.9*(MaxValue[i]-MinValue[i])
              + MinValue[i]
next i
R0:=Risk(k);
for t:=1 to 100,000 do
  for i:=1 to m do
       w[i]:= w0[i]
              + GaussRandom()*s
              *(MaxValue[i]-MinValue[i])
              R:=Risk(k);
  if R<R0 then R0=R; w0[i]=w[i] endif
  next i
next t
```

Fig. 9 The serial evolutionary algorithm.

For this, 10 performance time courses for different values of s are shown in Fig. 10. For each time step, all weights are mutated. Comparing this to Fig. 8, we observe a slight, but not one dramatic advantage. Also, since we have to evaluate the weights after each mutation, for one time step we have to evaluate 10 times, not once.

5. Discussion

The use of differential equation models in life science depends heavily on the approximation of the equation parameters to fit the observations.

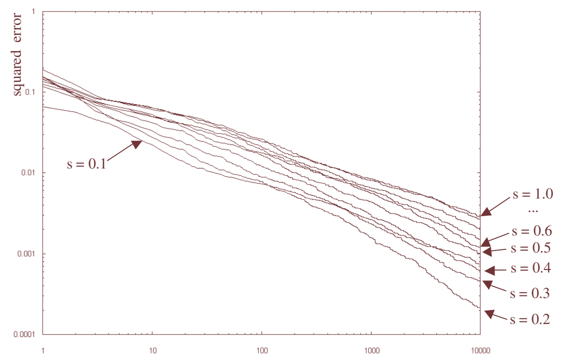


Fig. 10 Performance of the serial adaptive evolutionary algorithm

This paper introduces the problem for a small but fully functional model used in septic shock modeling and shows the advantages and disadvantages of several approaches.

It turns out that classical error correction learning is straight forward for fully known differential equation time courses, but it is not possible for the case of sparse observations. Here, random oriented optimum seeking algorithms should be preferred. According to the simulations, no significant performance difference between sequential and parallel adaptation schemes could be found.

6. References

[1] C. Chow, G. Clermont, R. Kumart, Z. Tawadrous, D. Gallo, B. Betten, G. Constantine, M. Fink, T. Billiar, Y. Vodovotz: "Quantitative Dynamics of the Acute Inflammatory Response in Shock States", *preprint* 2003 (personal communication)

[2] C. Chow: Development of a Dynamical Systems Model of Acute Inflammation, E. Neugebauer: 2nd workshop on complex systems: *Analysis in Shock and Trauma Research*, University of Cologne, 2003

[3] J. Paetz, R. Brause: "A Frequent Patterns Tree Approach for Rule Generation with Categorical Septic Shock Patient Data"; in: J. Crespo, V. Maojo, F. Martin, *Medical Data Analysis*, Springer Verlag Berlin Heidelberg 2001, pp.207-212

[4] B.Widrow, M. Hoff: "Adaptive switching circuits"; 1960 *IRE WESCON Convention Record*, New York: IRE, pp.96-104 (1960)

[5] A. Lapedes, R. Farber: "How Neural Nets Work"; Report LA-UR-88-418, Los Alamos Nat. Lab. 1988; and in Y.C. Lee (Ed.): *Evolution, Learning and Cognition*; World Scientific, Singapore, New Jersey, London 1988

[6] H. G. Schuster: *Deterministic Chaos: An Introduction*; Physik-Verlag, Weilheim 1984

[7] I. Rechenberg: *Evolutionsstrategie*; problemata frommann-holzboog, 1973