Script Dynamical Systems and autonomous agents

by

Ralf Der and J. Michael Herrmann

Part II

1 Behavior learning

Adaptivity is one of the main incentives for an artificial being striving for autonomy. Commonly adaptivity is seen as a tool for improving the performance of an agent in the completion of its tasks. Let us consider again an agent coupled to the world. At each instant of time t the agent sees sensor values $x \in \mathbf{R}^n$ and outputs an action y which changes the state of the world so that in the next instant of time the sensor values are changed. This can be modeled by the dynamical system

$$x_{n+1} = x_n + \theta W(x_n, y_n) \tag{1}$$

in the time discrete and

$$\dot{x} = W(x, y) \tag{2}$$

in the time continuous formulation, where the output y of the controller is a function of the sensor values x parameterized by the parameters c

$$y = K(x; c) \tag{3}$$

a generic example for K being given by the linear controller, (one-dimensional case)

$$y = ax + b$$

where we put $c = (a, b)$.

We ignored in the above formulation the possible existence of internal state with the corresponding update rule. This is not possible in many cases because of the existence of hidden variables. However in most cases this can be taken care of by enlarging the set of sensor values x by including the time derivatives and the like. We will therefore assume here again that the information contained in x is sufficient in order to characterize the state of the system.

As before we can use a more compact notation writing eq. 1 as

$$\dot{x} = F(x; c) \tag{4}$$

where $F(x;c) = W(x, K(x;c))$ and in shorthand

$$\dot{x} = F(x) \tag{5}$$

These equations describe the dynamics of the sensor-motor loop where x is the vector of sensor values and c the vector of the parameters of the controller. Each set of parameters c defines a behavior of the robot since for fixed c the output of the controller is a well defined function of the sensor values. We may therefore even call the parameter space the behavior space.

This is exemplified by Fig. ???.

Die Bilder fehlen noch!???

1.1 Formulation of the learning problem

The parameters can be tuned by hand form outside until the desired behavior of the robot is achieved. In an adaptive system the parameters are tuned by the agent itself according to some update rule defining the learning dynamics. Usually there is an error function E measuring the distance between the actual behavior and the target one. In general the error will be a function of both the actual state $x_n \in \mathbf{R}^n$ and the controller parameters $c_n \in \mathbf{R}^p$. Learning means to change c in order to minimize the error. This can be done in a systematic way by using gradient descent on the error function E, i.e. put

$$c_{n+1} = c_n - \varepsilon\theta \frac{\partial}{\partial c_n} E(x; c_n)$$

where ε is called the learning rate. In the time continuous formulation we get correspondingly

$$\dot{c} = -\varepsilon \frac{\partial}{\partial c} E(x;c)$$

The full dynamics including learning is now given by the extended dynamical system

$$\begin{aligned} \dot{x} &= F(x;c) \\ \dot{c} &= -\varepsilon G(x;c) \end{aligned} \tag{6}$$

where we introduced

$$G(x;c) = \frac{\partial}{\partial c} E(x;c)$$

which is the gradient of E. The formulation eq. 6 shows that now the parameters of the sensor motor loop obey a dynamics of their own, the time constant of this dynamics being given by the learning rate ε.

By introducing a joint state $z = (x, c)$ we may write eq. 6 more compactly as the dynamical system

$$\dot{z} = \Phi(z)$$

In principle we can also consider the parameters as internal states so that the learning dynamics is incorporated into the internal state dynamics as introduced earlier, see ???. However, these notations would hide the special role played by the parameters which we have seen to be responsible for qualitative changes in the system (bifurcation theory) and it is these effects which are the most interesting ones in the learning system. We will therefore in the following prefer the notation of eq. 6 in general.

1.1.1 Learning modes: Batch learning vs. on-line learning

Eq. 6 needs some further discussion. It expresses what is called on-line learning in that the learning step is executed immediately with the update of the sensor values. The point is that if the x dynamics for the current c runs into an attractor state $x^* = \phi(c)$ then the learning dynamics of c for large times

$$\dot{c} = -\varepsilon G(x^*, c)$$

is completely defined by this state alone so that other situations are not learned adequately. A remedy appears to be to rerun the full dynamics with different initial conditions in order that the (x, c) space is covered sufficiently dense until convergence to the target behavior is achieved. Howevr this needs further concretization.

In order to discuss this point we study the linear one-dimensional case with a linear controller $K(x; c) = ax + b$ and a world function $W(x, y) = y$ so that the dynamics of the sensor values is

$$\dot{x} = ax + b$$

We use the hypothetical error (which will be recovered in the examples below)

$$E = (ax + b - Ax - B)^2 = D^2 \tag{7}$$

which of course is zero for $a = A$ and $b = B$ for any value of x. However for any x given there is a one parameter family of solutions for $E = 0$ and it is this point which makes itself felt in many instances as will be seen in the following. The point is that the gradient of the error which defines the learning step is obtained for a specific x and so it not necessarily points into the direction of the globally (for all x) optimal solution.

Using the above error the joint dynamics is

$$\begin{aligned} \dot{x} &= ax + b \\ \dot{a} &= -\varepsilon \left((a - A) x + b - B \right) x \\ \dot{b} &= -\varepsilon \left((a - A) x + b - B \right) \end{aligned} \tag{8}$$

With $b = B = 0$ we obtain trivially

$$\begin{aligned} \dot{x} &= ax \\ \dot{a} &= -\varepsilon x^2 (a - A) \end{aligned}$$

so that indeed $a \to A$ for large times independently on the value of x. Now let us include the b dynamics again and consider the case $A < 0$ and $a < 0$ so that the x dynamics converges exponentially to $x = -b/a$. If the time scales are well separated which means ε is sufficiently small we may assume that the x dynamics is always converged, i.e. we may replace $x(t) = -b(t)/a(t)$ in the learning dynamics to obtain

$$\begin{aligned} \dot{a} &= \varepsilon \frac{b}{a} \left(A \frac{b}{a} - B \right) \\ \dot{b} &= -\varepsilon \left(A \frac{b}{a} - B \right) \end{aligned} \tag{9}$$

which now so to say is autonomous, i.e. is decoupled from the x dynamics. (This procedure is an example of the elimination of fast variables to be discussed in more detail in the Chapter on self-organization ???)

Now b converges towards aB/A. The behavior of a is discussed by noting that $\dot{a}a + \dot{b}b = 0$ in the case considered so that $a^2 + b^2$ is constant over time. Then because of $b \to aB/A$ we find that for $a < 0$ and $A < 0$

$$a(\infty) = A \sqrt{\frac{a^2(t_0) + b^2(t_0)}{A^2 + B^2}}$$

which means that the asymptotic value $a(\infty)$ of a depends on the initial values of both a and b.

The result that a does not converge towards A has an easy geometrical explanation. We introduce a vector $\vec{x} = (x, 1)^T$, $\vec{C} = (A, B)^T$, and $\vec{c} = (a, b)^T$ so that

$$E = \left(\vec{G} \cdot \vec{x} \right)^2$$

where $\vec{G} = \vec{a} - \vec{A}$ in the case of eq. 7. Obviously the component of $\vec{a}$ perpendicular to $\vec{x}$ is not affected by the learning procedure so that it keeps

its initial value. Hence in order that $a \to A$ as desired for the global optimum we must restart the system repeatedly in the hope that the learning part of the full dynamics sees enough different values of x. Each new x leaves a different component invariant so that different x destroy gradually each others orthogonal components and in the end full convergence is achieved. This scenario is formalized in the so called batch learning to be discussed now.

Batch Learning By the restarting scenario values of x will occurr with a certain probability distribution $P(x)$. Batch learning essentially corresponds to carrying out the averaging over a representative set of states (training examples) first and then gradient descent the averaged E. The typical formulation of this paradigm is the following. We assume there is a set T of x values (training set) with the probabilities $P(x)$ that this x occurrs in the learning step. Then the average error is

$$\overline{E}(c) = \sum_{x \in T} P(x) E(x; c) \tag{10}$$

The gradient G is now only a function of c and we write

$$G(c) = \frac{\partial}{\partial c} \overline{E}(c) \tag{11}$$

Noting that averaging over the training set and taking the derivative can be interchanged we may also obtain $G(c)$ as the average over the individual updates, i.e. consider

$$G(c) = \sum_{x \in T} P(x) \frac{\partial}{\partial c} E(x; c) = \sum_{x \in T} P(x) G(x; c) \tag{12}$$

The name batch learning results form this fact, namely that each update Δc of c according to eq. [**?**] is batched (stored) and only after many examples from the training set are considered the updates are summed up and the learnig step carried out. Note that the averaging over P occurrs automatically in this scenario since the number of cases to a certain value of x is proportional to $P(x)$.

Whatever the way to the average gradient is, the c are learned offline then according to

$$\dot{c} = -\varepsilon G(c) \tag{13}$$

In the above simple example we find

$$\begin{aligned}
\overline{E}(c) &= \sum_{x\in T} P(x)(ax+b-Ax-B)^2 \qquad (14)\\
&= (a-A)^2\overline{x^2} + 2(a-A)(b-B)\overline{x} + (b-B)^2 \qquad (15)\\
&= (a-A)^2\sigma^2 + ((a-A)\overline{x}+b-B)^2 \qquad (16)
\end{aligned}$$

where

$$\sigma^2 = \overline{\left(x^2 - \overline{x}^2\right)}$$

is the variance. We find for $\overline{x}=0$ trivially that $b=B$ and $A=a$ is the global minimum. For the $\overline{x}\neq 0$ case we consider the learning dynamics now is (absorbing a factor 2 into ε)

$$\begin{aligned}
\dot{a} &= -\varepsilon\left((a-A)\overline{x^2} + (b-B)\overline{x}\right)\\
\dot{b} &= -\varepsilon\left((a-A)\overline{x} + (b-B)\right)
\end{aligned} \qquad (17)$$

Obviously, the averaging guarantees the convergence to the global optimum. The stability analysis shows that there are two non-negative eigenvalues the smaller one of which is

$$\lambda_0 = -\frac{\sigma^2}{\left(\overline{x}^2+1\right)} \qquad (18)$$

meaning that there is a subspace which is invariant under the dynamics for $\sigma=0$ which decays as soon as $\sigma^2>0$ the variance σ^2 defining the time constant for the learning process for small values of σ^2 (exercise).

1.1.2 Stochastic gradient descent

The connection between the batch and on-line learning modes is given by the stochastic gradient scenario. When doing on-line learning, i.e. using eq. 6 with repeated restarts the question of the convergence of the procedure is still open. The connection between the two approaches is ruled by the method of the so called stochastic gradient descent. Let us consider eq. 12 which gives the relation between the current value of the gradient and the average one. We write

$$G(x;c) = G(c) + \delta G \qquad (19)$$

so that $G(c)$ may be called the systematic and δG the stochastic part of the gradient. The latter averages out, i.e. $\overline{\delta G(x;c)} = 0$, in the averaging procedure of eq. 12. The question is what its part is in the case of the combined dynamics of eq. 6. Let us write the latter as

$$\begin{aligned} \dot{x} &= F(x;c) \\ \dot{c} &= -\varepsilon \left(G(c) + \delta G(x;c) \right) \end{aligned} \tag{20}$$

and ask abotut the role of the δG term. Of course as long as $\varepsilon > 0$ is constant the influence of the term will not vanish. Instead it will cause fluctuations of the update step around the systematic part given by $G(c)$. The trick one uses in order to damp these fluctuations is to shrink ε gradually down to zero so that the fluctuations disappear eventually. The question is about the regime of this shrinking or cooling down as it is called of ε. Obviously if it is done too fast than the learning dynamics does not see enough different values of x with the consequences given above. On the other hand if the cooling is too slow one never comes to the end.

Fortunately there is a theorem according to Robins and Monroe which clarifies the case. In order to formulate the theorem we consider the parameter dynamics part of eq. 20 and note that in the restarting scenario what happens is that this part sees in the course of time x values with a certain probability $P(x)$. If the restarting scenario is chosen conveniently (choosing widely different starting values for x while running the dynamics only a short interval of time so that there is no convergence to any FP) we may assume that the learning dynamics (time discrete notation)

$$\Delta c = -\varepsilon G(x;c)$$

may be considered separately where we assume that the values of x are drawn randomly with the probability distribution $P(x)$. Under convenient conditions for this distribution one may use the Robins Monroe theorem to state that the values of c converge towards a minimum of $\overline{E}(c)$ if only the parameter ε fulfilles the condition

$$\varepsilon \sim t^{-\alpha}$$

where $0 < \alpha < 1/2$.

This result gives us the backup to use the on-line learning scenario with a convenient restarting regime for the system which essentially means that one has to use a sufficiently large range of starting values $x(0)$ and stop the system dynamics long before the pertinent fixed points are reached. We will make these statements more explicite in the examples considered below.

1.2 Learning tasks

We are now going to formulate two of the most common learning tasks with further tasks to be formulated below.

1.2.1 Supervised learning

In supervised learning the training of the learner involves a set

$$\Xi = \{(x, Y) \,|x \in T\} \tag{21}$$

of training examples given by pairs (x, Y) where $Y \in \mathbf{R}^m$ is provided by some trainer (supervisor). We may assume that there is a function $L : \mathbf{R}^n \to \mathbf{R}^m$ so that $Y = L(x)$ and L describes the behavior of the trainer. The error between the present and the target output of the controller may be measured by the Euclidean distance

$$d(y, Y) = \|y - Y\|^2 = \sum_{i=1}^{m} (y_i - Y_i)^2 \tag{22}$$

This error is a function of both x and c as is seen from noting that

$$E(x; c) = d(y, Y) = \|K(x; c) - L(x)\|^2 \tag{23}$$

The aim of learning is to adapt the function $K(x; c)$ so that the difference

$$E(x; c) = \|y - Y\|^2 = \|K(x; c) - L(x)\|^2$$

is minimized. Hence this kind of learning corresponds essentially to a function approxmation task.

1.2.2 Goal oriented learning – How to approach a target state

Another common learning task is given by the case that the ´robot is to reach a situation with prescribed sensor values X. The Euclidean distance

$$d(x, X) = \|x - X\|^2 = \sum_{i=1}^{n} (x_i - X_i)^2 \tag{24}$$

is a measure for the deviation of the current state to the target one wich is the information about the distance we still have to the goal state. With the controller given the robot will move so that the sensor values change.

Consider (in a time discrete formulation which easily translates into the continuous one)

$$E(x;c) = \sum_{k=0}^{K} d(x_{n+k}, X) \tag{25}$$

where x_{n+k} is the state after k time steps starting in the current state which is $x_n = x$. We follow the evolution of the distance over K time steps into the future. We may also use a discount rate $0 < \gamma < 1$ and put

$$E(x;c) = \sum_{k=0}^{\infty} \gamma^k d(x_{n+k}, X) \tag{26}$$

which damps events in the future exponentially. The error is the smaller the more rapidly the controller approaches the robot to the target state. It is therefore a function of the parameters of the controller and can thus be used to improve the controller by gradient descent.

In either case the learning can be done in both the on-line or batch mode.

1.2.3 Autonomous learning

In a scenario called autonomous learning there is neither a trainer giving the target output for the controller nor is there a goal state defined. Instead the agent is thought of following some internal principle and to adapt itself so that it is in concordance with this principle as much of possible. An example for such a principle is formulated as the desire of the agent to understand its own behavior in the world.

Autonomous learning will play a particular role in this work and we will reserve an extra chapter for this (see below).

1.3 Generic example

Let us consider by way of example a wheeled robot which is to move with a constant forward velocity v, the controller output y being the turn velocity of the robot in physical space. The robot has two infrared sensors with readings x_1, x_2 looking into the forward direction. The task the controller is to learn consists in keeping the ball in the middle between the two sensors which means nothing else but to keep the difference $x = x_1 - x_2$ at zero. Then we may put the error at the current time as

$$E(x) = (x_1 - x_2)^2 = x^2$$

We may simplify the task a little and assume that as a result of a preprocessing the controller sees the difference coordinate x right away. In a linear approach the controller is put as

$$y = K(x)$$

where

$$K(x) = K(x;c) = ax + b \tag{27}$$

where $c = (a, b)$. If the ball is not moving and $v = 0$ the change in sensor value is a direct function of the state x itself and the controller output y (see the discussion in ???) so that in this case the pertinent dynamical system describing the time evolution of the sensor values is

$$\dot{x} = \Phi(x, y)$$

so that $F(x;c) = \Phi(x, y(ax + b))$. If $v \neq 0$ the robot pushes the ball while moving and the relative position of the ball to the sesors is influenced by a number of more or less random factors. We may lump these into a random function $\xi(t)$ and write

$$\dot{x} = \Phi(x, y) + \xi(t) \tag{28}$$

and we asssume that in the average over many trials or over time in a single trial the perturbation ξ is equal to zero.

We may make the world function Φ a little more explicite. We model the fact that the ball will leave the region of good control (by rolling to the side) by a term which increases the sensor values rapidly once $|x|$ becomes large. We put

$$\dot{x} = \gamma x^3 + K(x) + \xi(t) \tag{29}$$

where γ is a hardware constant which may be reset to $\gamma = 1$ by a rescaling of time.

The task now consisits in finding a convenient learning procedure so that the robot learns to keep the ball in front of itself.

In practice it is not so easy to find the error as a function of c. One possibility is to learn a world model first which means a function $M(x, y)$ which approaches $W(x, y)$. Then we may generate approximate trajectories by the model dynamics

$$\dot{x}^P = M\left(x^P\right)$$

where $M\left(x^P\right)$ is again shorthand for $M\left(x^P, K\left(x^P\right)\right)$ using x_n as the starting state. Another possibility is to learn the error function in a number of trials with different parameter sets c. We will discuss these procedures in the examples below.

1.3.1 Supervised learning

In supervised learning we need a trainer which in each instance of time yields the target ouput of the controller as a function of x. Let us assume the trainer proposes a linear control strategy so that he is described by the function

$$L(x) = Ax + B$$

with A and B fixed. Then the error is

$$E = (K(x) - L(x))^2 = ((a - A)x + b - B)^2$$

which is just the error function studied above, the full dynamics being given by eq. 8. We know from the above discussion that the learning converges towards $a = A$ and $b = B$ in the batch scenario provided the learner sees not only a single value of x. The distribution of x does not matter (it affects only the convergence time) so that the stochastic gradient descent procedure will converge as well if the system is restarted repeatedly. In practice the cooling of ε can be done much more rapidly as in the strict sense.

1.3.2 Goal oriented learning – approaching a target state

Let us now assume that we prescribe the goal $X = 0$ so that ther error over one step (i.e. $K = 1$ in eq. 25) of length τ now is

$$E(x) = x^2 + x^2(t + \tau)$$

where

$$E(x) = x^2 + (x + \tau\dot{x})^2 = x^2 + (x + \tau(ax + b))^2$$

in the small τ case and the full dynamics becomes

$$\begin{aligned} \dot{x} &= ax + b \\ \dot{a} &= -\rho(x + \tau\dot{x})x \\ \dot{b} &= -\rho(x + \tau\dot{x}) \end{aligned}$$

where $\rho = \varepsilon\tau$ and we can write the r.h.s of the parameter dynamics also as

$$\begin{aligned} (x + \tau\dot{x})x &= \left(1 + \frac{\tau}{2}\partial_t\right)x^2 \\ (x + \tau\dot{x}) &= (1 + \tau\partial_t)x \end{aligned}$$

Hence as long as in the first equationn the time derivative of x^2 is smaller than x^2 itself than $\dot{a} < 0$ so that a decreases and will reach negative values

after sufficient long time. Once $a < 0$ we may assume again (see the discussion above) the slaving condition for x to be fulfilled, i.e. that $x = -b/a$ to obtain the learning dynamics

$$\begin{aligned} \dot{a} &= -\rho \frac{b^2}{a^2} \\ \dot{b} &= \rho \frac{b}{a} \end{aligned}$$

which shows that $b \to 0$ since $a < 0$. The dynamics has a first integral

$$a^2 + b^2 = \text{const}$$

so that for $a(t_0) << 0$ the value of a decays so that

$$a(\infty) = -\sqrt{a^2(t_0) + b^2(t_0)}$$

In case $a > 0$ initially x diverges so that after some time $\dot{x} = ax$ and $(x + \tau\dot{x})\, x = x^2 (1 + \tau a)$ in a good approximation. This approximation prevails as long as $a > 0$ since x will still further increase monotonuously. However $\dot{a} < 0$ in this regime so that a is driven towards negative values. This scenario for a is made complete by noting that $\dot{b} = -\rho(1 + \tau a)\, x$ so that b changes more slowly than ax in the x dynamics.

Surprisingly there is still another regime of the entire system found by expanding around $x = 0$, $a = 0$ and $b = 0$. We find

$$\begin{aligned} \dot{x} &= b \\ \dot{a} &= 0 \\ \dot{b} &= -\rho(x + \tau b) \end{aligned}$$

and obtain the eigenvalues of the x, b system as

$$\lambda_{\pm} = -\frac{1}{2}\rho\tau \pm i\sqrt{\rho}$$

Hence x decays to zero in a damped oscillation, so that the aim of the learning is also achieved however in a somewhat unexpected scenario. The difference to the former scenario is that we do not obtain a controller with a fixed set of parameters in this regime but instead the controller achieves its function by a permanent relearning of its parameters, b in this case. This is strange but will prove helpful in situation where decisions are to be made between different strategies.

1.3.3 The role of the parameterization

It is interesting that the behavior of the full dynamics is in an essential way dependent on the parametrization of the controller. In order to illustrate this point we consider the case that we write

$$K(x) = a(x - b)$$

where b now is a bias on the sensor value which is to be adapted by the controller. The error now reads

$$E(x) = x^2 + (x + \tau\dot{x})^2 = x^2 + (x + \tau a(x - b))$$

and the dynamical equations of the learning system are

$$\begin{aligned} \dot{x} &= ax - ab \\ \dot{a} &= -\rho(x + \tau\dot{x})(x - b) \\ \dot{b} &= \rho(x + \tau\dot{x})a \end{aligned}$$

or

$$\begin{aligned} \dot{x} &= ax - ab \\ \dot{a} &= -\rho(x + \tau a(x - b))(x - b) \\ \dot{b} &= \rho(x + \tau a(x - b))a \end{aligned}$$

in leading order of τ we obtain

$$\begin{aligned} \dot{x} &= a(x - b) \\ \dot{a} &= -\rho x(x - b) \\ \dot{b} &= \rho x a \end{aligned}$$

Assuming $a < 0$ again we approximate $x = b$ and

$$\begin{aligned} \dot{a} &= 0 \\ \dot{b} &= \rho a b \end{aligned}$$

which guarantees the convergence $b \to 0$ since $a < 0$. So everything is as in the case discussed above.

We also have the damped oscillation regime here as is seen from the stabilitiy analysis around the FP $x = 0$ and $a = 0$ for arbitrary b differnt from zero. We write in leading order

$$\begin{aligned} \dot{x} &= -ab \\ \dot{a} &= \varepsilon\tau b x - \varepsilon b^2\tau^2 a \\ \dot{b} &= 0 \end{aligned} \tag{30}$$

the characteristic matrix for the x, a part being

$$\begin{pmatrix} 0 & -b \\ \varepsilon\rho & -\varepsilon\rho^2 \end{pmatrix}$$

where $\rho = \tau b$ with eigenvalues

$$\lambda_{\pm} = -\frac{1}{2}\rho^2\varepsilon \pm \frac{1}{2}\sqrt{\rho^4\varepsilon^2 - 4\rho b\varepsilon}$$

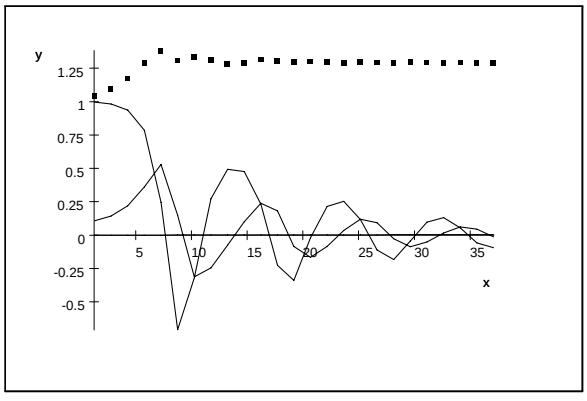
which for small ε read

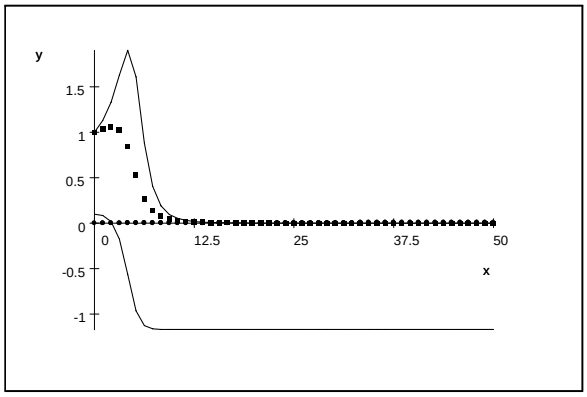
$$\lambda_{\pm} = -\frac{1}{2}\rho^2\varepsilon \pm i\sqrt{\varepsilon\tau b^2}$$

which is a damped oscillation. Using a time scale ε^{-1} we see that the damping per oscillation decreases with decreasing ε.

The oscillations are prominent in the full system if the starting values for x and b are of the same sign. Then we obtain a **limit cycle oscillation** at rather large amplitudes for the state variable x.

$$\begin{aligned} x' &= ax - ab \\ a' &= bx\tau - x^2\tau + 3abx\tau^2 - ab\tau^2 - 2ax^2\tau^2 \\ b' &= x\tau a + \frac{3}{2}x\tau^2 a - \tau^2 ab \qquad (31) \\ x(0) &= 1 \qquad (32) \\ a(0) &= 0.1 \qquad (33) \\ b(0) &= 1 \qquad (34) \end{aligned}$$





Learning the ball pushing task with one-step ahead error and starting values $x(0) = 1$, $c_0(0) = 1$, and $c_1(0) = 0.1$. There was a nonlinearity $0.1x^3$ added to the dynamics.

However these effects are more or less subtleties of the linear control task. If we add small nonlinearities like in the ball pushing task we find a rapid and systematic convergence towards the globally optimal control behavior.

1.3.4 The role of the time horizon

We might attribute the above effects of nontrivial convergence to the greedy nature of our learning algorithm. The idea then might be that it is the one step procedure of following the deviation of the state $x(t)$ from the target value $X = 0$ over one step only. However in the present very simple example we can extend the time horizon up to infintiy. Let us consider the future deviations of the state from the target one integrated over a window function onto infinity, i.e.

$$E(x) = \int_0^\infty W(\tau)\, x^2(t+\tau)\, d\tau$$

where we use the normalized window function

$$W(\tau) = 2\lambda e^{-2\lambda\tau}$$

with λ sufficiently large so that the integral exists. The formula now become a little more involved, however the main conclusions remain the same. In particular we again obtain the damped oscillatory regime which is easily seen from the fact that it involves the $a\tau << 1$ limit which means replacing the $\exp(a\tau) = 1 + a\tau$ so that the stability analysis leads to the same results as the one given above.

1.3.5 Learning a world model

It is easy int the above system to find the world model. In fact the latter can never be expected to reproduce the effects contained in $\xi(t)$. The update in eq. 28 may be viewed as the sum of a systematic part Φ and the random one ξ. Then all we can expect is to model the systematic part. In order to find a learning rule for the former we need a parameterized expression for the world model. Let us use a linear expression for the time being, i.e. put

$$M(x,y) = u_1 x + u_2 y + u_0$$

where u is the set of model parameters. Let us assume we sample the trajectory generated by eq. 28 in time intervals θ. A convenient error for learning M is

$$F(x,y) = (\dot{x}_{emp} - M(x,y))^2$$

where we have to use the empirical derivative

$$\dot{x}_{emp} = \frac{x(t+\theta) - x(t)}{\theta}$$

We use gradient descent for the learning of the model

$$\Delta u_i = -\eta \frac{\partial}{\partial u_i} F$$

The learning has to be repeated for a sufficiently large sample of values x and y. This requires that during the learning phase the system must be kept in an explorative mode of behavior.

We can analyse the learnig behavior in some detail using the random character of the perturbation (noise) ξ. In fact we may write more explicitly

$$-\frac{\partial}{\partial u_i} F = (\dot{x}_{emp} - M(x,y))\, z_i$$

where $z_0 = 1$, $z_1 = x$, $z_2 = y$. Using eq. 28 we find that the gradient is the sum of a systematic and a random part

$$-\frac{\partial}{\partial u_i} F = (\Phi(x,y) - M(x,y))\, z_i + \xi z_i$$

so that in the average

$$-\frac{\partial}{\partial u_i} F = (\Phi(x,y) - M(x,y))\, z_i$$

Hence by the learning procedure the model tries to fit the systematic part of the dynamical system, the quality of the fit depending on the structure of the model. In the present example we have chosen a linear model whereas Φ will be rather nonlinear in general. However if we restrict to a small region in sensor space the linear model will do sufficiently well.

1.3.6 Pseudolinear functions

In many cases the linear ansatz is too simple. This is in particular the case for the controller if the latter must not prescribe too large values for the actuators. In this case it is advisable to use a so called squashing functions. If for instance the output of the controller is to be kept in the region $-1 < y < 1$ one may use the tanh function, i.e. put

$$K(x) = \tanh(ax + b)$$

instead of eq. 27. The tanh function obeys the nice property that

$$\frac{d}{ds}\tanh(s) = 1 - \tanh^2(s)$$

so that the learning rule involving this kind of pseudononlinearity stay the same apart from an additional factor $1 - \tanh^2()$ which multiplies the learning rate. Thus learning is slower in the region where the output of the controller is high. This kind of pseudolinearity is one of the ingredients of neural networks to be discussed below so that we do not dwell on the subject here.

2 Realization of adaptive controller by neural networks

In practice adaptive controllers are often realized by a neural network since there are detailed learning rules once the error function is given. We therefore will in the following section consider neural networks as realizations of parameterized controllers. The fundamental building stone of any neural network is the formal neuron which is a (largely simplified) model of the true biological neuron. We start by giving the standard model and will see later on how neurons are combined to form neural networks.

2.1 Mathematical model of a single neuron

In the most simple setting the neuron is considered as a parameterized input-output device. The input is represented by a vector $x \in \mathbf{R}^n$ and the output of a neuron i is written as

$$y_i = g(z_i) = f_i(\mathbf{x})$$

where $f : \mathbf{R}^n \to \mathbf{R}^1$ is the so called transfer function, z is the post synaptic potential which is the weighted sum over the inputs

$$z_i = \mathbf{w}_i \cdot \mathbf{x} = \sum_{j=1}^{n} w_{ij} x_j$$

the w_{ij} being the synaptic efficacies of the i-th neuron connected to neuron j. The function $g : \mathbf{R}^1 \to \mathbf{R}^1$ is the activation or squashing function, typically a sigmoid

$$g(z) = \frac{1}{1 + e^{-\beta z}}$$

which monotonously increases from 0 to 1 as z increases from large negative to positive values. Another choice is the tanh function as discussed above ???.

In the learning rules one often needs the derivative of the sigmoid with respect to z. It is easily seen that

$$\frac{\partial}{\partial z} g(z) = \beta g(z)(1 - g(z))$$

where we remember that $g(1-g)$ is the logistic map.

2.2 Neuronal learning

Neuron i maps its inputs to the output the map being parameterized by the weight vector $\mathbf{w}_i$. The weights can be adapted in the following supervised learning scenario: Given a set of input-output mappings (provided by a trainer) find $\mathbf{w}_i$ such that the neuron reproduces these pairs as close as possible. In an on-line learning scenario in each learning step the neuron sees an input $\mathbf{x}$ together with the target output Y provided by the trainer. The error the neuron i makes is

$$E_i = \frac{1}{2}(Y - y_i)^2 \tag{35}$$

where $y_i = g(\mathbf{w}_i \cdot \mathbf{x})$ ist the current output of the neuron. Gradient descent yields the update

$$\Delta w_{ij} = -\varepsilon d_i x_j \tag{36}$$

for the synaptic weights. Viewing the error signal

$$d_i = g'(z_i)(Y - y) \tag{37}$$

where $z_i = \mathbf{w}_i \cdot \mathbf{x}$ as the error activity at the output of the neuron i we may interprete the synaptic change in terms of the Hebb rule which enhances synaptic strenght if activities on the pre- and postsynaptic side are both high.

3 Feed-forward networks

Neurons can be combined to neural networks in order to represent more complicated input output mappings. In a typical feed forward architecture the net realizes a function $f: \mathbf{R}^n \to \mathbf{R}^m$, i.e. it maps inputs $x \in \mathbf{R}^n$ on outputs $\mathbf{y} \in \mathbf{R}^m$. The network is organized as a sequence of layers, the neuron outputs of a layer forming the inputs to the neurons of the subsequent layer. A network of N layers consists of the input layer where $\mathbf{x}$ is fed in, a number $M \geq 0$ of hidden layers and an output layer of m neurons. Numbering the layers from the input layer ($k = 0$) to the output layer ($k = N$) the rule for feeding the input information through the network is given by the iterative rule (identical gain functions for all neurons)

$$y_i^{(k)} = g\left(\mathbf{w}_i^{(k)} \cdot \mathbf{y}^{(k-1)}\right), \qquad k = 1, 2, \ldots, N \tag{38}$$

starting with $\mathbf{y}^{(0)} = \mathbf{x}$, $\mathbf{y}^{(k)} = \left(y_1^{(k)}, y_2^{(k)}, \ldots\right)$ being the vector of outputs of the neurons and $\mathbf{w}_i^{(k)}$ is the synaptic vector of neuron i in layer k.

There is a theorem stating the universality of neural networks as function approximators: Already a network with only one hidden layer (with a sufficient number of neurons) is able of representing any function $\mathbf{F}: \mathbf{R}^n \to \mathbf{R}^m$ with arbitrary accuracy. A ready explanation is that the neurons of the hidden layer can be organized into groups so that the collective receptive field of a group covers regions in input space mapping to about the same value of the target function $\mathbf{F}$. Then, all the output layer has to do is to weigh the contributions of the groups in order to produce the correct output $\mathbf{Y} = \mathbf{F}(\mathbf{x})$ to any input $\mathbf{x}$.

This function approximation can be learned by gradient descending the error $E = (\mathbf{Y} - \mathbf{y})^2$. This can be transformed into a systematic procedure for the indivdual updates of the synaptic weights

$$\Delta w_{ij}^{(k)} = -\varepsilon d_i^{(k)} y_j^{(k-1)} \tag{39}$$

which is Hebb like again. The error activities are obtained in an iterative way

$$d_i^{(k)} = g'(z_i) \sum_l d_l^{(k+1)} w_{li}^{(k+1)} \tag{40}$$

so that the error signal for neuron i in layer k is produced by the weighted sum of the error signals in the layer $k+1$, the iteration starting with the error signals at the output neurons, cf. eq. (37). Thus the error is propagated backwards thtough the network which is the reason why this famous algorithm is called error back-propagation algorithm.

4 Recurrent networks

The feed forward networks serve well as function approximators. However in many application mainly in time series prediciton the output of the network is to depend on inputs seen previously. This can be achieved with additional inputs into the network of earlier events. A more elegant way consists in recurrent networks which internally built up a memory of the past by time delayed recurrencies.

A standard RNN type network is obtained by equipping a FFN with feed back loops feeding a time delayed copy of the output of a layer back to the layer itself. In this scenario inputs are presented to the network at times $t = 0, 1, \ldots$ The iterative rule (38) for evaluating the output of the network is now

$$y_{t,i}^{(k)} = g\left(\mathbf{w}_i^{(k)} \cdot \mathbf{y}_t^{(k-1)} + \mathbf{v}_i^{(k)} \cdot \mathbf{y}_{t-1}^{(k)}\right), \qquad k = 1, 2, \ldots, N$$

where $\mathbf{y}_t^{(0)} = \mathbf{x}_t$ is the current input into the network, $\mathbf{y}_t^{(k)}$ the vector of current activations of the layer k, and $\mathbf{v}$ is the set of weights controlling the feed back of the activations of the previous time step.

The network is now a discrete-time dynamical system driven by the inputs $\mathbf{x}_t$. Depending on the weights the network can develop the full range of dynamical behaviors like fixed point attractors over limit cycles to chaos. Again the important point is that the weigths can be learned such that the network reproduces a target dynamics. It is this property which makes the

recurrent networks valuable tools for time series prediction, system identification and signal processing.

5 Competitive learning

So far we have considered supervised learning of input-output mappings. Another performance of networks consists in clustering and feature extraction which are also of biological relevance. These properties can be learned in an unsupervised fashion if they are based on the statistical properties of the data alone.

In this scenario we have a set T of input vectors $\mathbf{x} \in T$ and an ensemble of neurons $i = 1, 2, \ldots, K$. In the engineering domain the activation of a neuron i is given in terms of the Euclidean distance between the synaptic vector $\mathbf{w}_i$ and the input vector $\mathbf{x}$

$$\delta_i(x) = (\mathbf{x} - \mathbf{w}_i)^2 = \sum_{j=1}^{n} (x_j - w_{ij})^2$$

The distance may be interpreted as the error the neuron makes in trying to represent (match) the input. There is always a best matching neuron (choose one by chance if there are several) which is called the winner (of the competition for the best match). Neuron $i$is the winner if

$$\delta_i(x) \leq \delta_j(x) \quad \forall j$$

Learning is the competition for improving the match with the inputs. In a learning step input $\mathbf{x}$ is presented and the winner chosen. The winner learns according to

$$\Delta \mathbf{w}_i = -\varepsilon(\mathbf{w}_i - \mathbf{x}), \qquad i: \quad \delta_i(x) \leq \delta_j(x) \quad \forall j$$

which means that the winner moves its synaptic vector a little towards the present input which improves its representation of the inputs close to $\mathbf{x}$. As a result of this greedy strategy there are neurons representing no inputs at all. Of the remaining ones each neuron represents about the same number of inputs from the set T. In this way the active group of neurons has learned to represent the data set T in a most effective way.

In a more formal understanding the set of synaptic vectors realizes a Voronoi tesselation of the input space, each cell V_i of the tesselation being

defined as the set of vectors $\mathbf{x} \in \mathbf{R}^n$ which are closer to $\mathbf{w}_i$ than to any other of the $\mathbf{w}_j$ where $j \neq i$

$$V_i = \{\mathbf{x} | \delta_i(\mathbf{x}) < \delta_j(\mathbf{x}) \quad \forall j \neq i\}$$

Competitive learning then produces a tesselation such that the number of inputs $\mathbf{x} \in T$ is about the same in each of the active cells. There are several generalizations of the competitive learning which manage to avoid the creation of passive neurons the common idea being to relaxe the competition so that the algorithm becomes less greedy (soft competitive learning procedures).

6 Self-organizing feature maps

Feature maps form a major processing element of real brains. Features are abstractions from high-dimensional noisy data. These abstractions are many to one, usually there is a bunch of input data inducing the same feature so that the feature represents the main information in this set of inputs. Feature maps are serve the mapping of the input space onto a set of features. Such maps are of particular usefulness for further processing steps if the mapping is topographic meaning that (i) the features are represented in a space with a topology and (ii) the map preserves the neighborhood relationships of the input space.

In the ANN domain the features are represented by neuron occupying the sites of a lattice which constitutes the feature space. The target feature (neuron) of a given input is the one with the best match in the sense explained above.