Formal modelling in Cognitive Science 1: Maths for Neural and Connectionist Modelling

Mark C. W. van Rossum
mvanross@inf.ed.ac.uk

January 2, 2007

Preface

These lecture notes are for a course of fifteen lectures is designed for students in the first year of their Cognitive Science degree. It is part of the full course 'Formal modelling in Cognitive Science 1'. It covers the mathematical tools commonly used to study neural models, cognitive science and related areas. Comments to these lecture notes are always welcome at mvanross@inf.ed.ac.uk.

Literature There are many mathematics textbooks at intermediate level: e.g. [Boas, 1966]. A good book that we use, is Greenberg's [Greenberg, 1998]. It is a bit expensive, but a good investment.

By now the internet is also a good resource (with the usual caveats) `http://mathworld.wolfram.com` contains an encyclopedia of mathematics.

A good introductory book in neural computation is Trappenberg's [Trappenberg, 2002]. A more formal but very good book on neural networks: [Hertz et al., 1991]. See lecture notes Neural Computation on `http://homepages.inf.ed.ac.uk/mvanross` for more references and applications to neural systems and literature relevant to neural modelling.

Software Numerical calculations can be done using Matlab (installed on DICE). Octave is a decent, free clone of Matlab (older version on DICE). A recent alternative is 'R', which has a neat structure, but is not compatible with Matlab.

Programs that deal with symbolic math are Maple (xmaple on DICE), maxima (free), and Mathematica.

Contents

1 Linear Algebra: Vectors

1.1 Vectors

In some cases a single number is enough to describe a system, in many other cases a collection of numbers is necessary. When the individual variables all live in the same space we call it a vector. Vector variables are commonly denoted bold-face or, when hand-written, with a vector arrow. Example:

$$\mathbf{v} = \vec{v} = (v_1, v_2, v_3, v_4)$$

This is a four-dimensional vector. An example of a vector in the physical world is the position in 3 dimensions; velocity is another example. Computer-scientists have often a more loose definition of vector, namely as just a list of items or numbers, not necessarily related to each other.

Addition and subtraction are done by component: suppose $\mathbf{v} = (v_1, v_2)$, $\mathbf{w} = (w_1, w_2)$. Now $\mathbf{v}+\mathbf{w} = (v_1+w_1, v_2+w_2)$ and $\mathbf{v}-\mathbf{w} = (v_1-w_1, v_2-w_2)$, see Fig. 1.

The distinguish numbers from vectors we use the term SCALAR to indicate just a single number. Scalar multiplication with a vector is defined as $\alpha\mathbf{v} = (\alpha v_1, \alpha v_2, \alpha v_3, \alpha v_4)$ and thus scales the vector. If we vary α over all real numbers and we plot all points $\alpha\mathbf{v}$, we obtain a line through the origin.

We just described a vector using n (Cartesian) coordinates. Alternatively, one can define a vector by its length and a direction (specified by $n-1$ angles for a n-dimensional vector).

Relevance to cognition

> Many physical quantities are vectors. There are also applications to cognitive science: the visual input at a particular instant can be described as a vector. The number of dimension equals the number of pixels (ganglion cells in the retina). For human retina there are about 1 million of these, and the input dimension is therefor 1 million. One million numbers are necessary to describe the image (we do not consider colour vision here, which would triple this number). Note, that the geometry of the retina itself is two-dimensional, but that is not important for the signal processing.
>
> In sensory systems the input space is usually very high dimensional. This allows for very rich set of input patterns, but for the scientist it complicates study of sensory systems and their input-output transformation.

1.2 Distance

There are various ways to calculate the lengths of vectors (also called norm) and distances between vectors. The most common one is the Euclidean norm. The Eu-

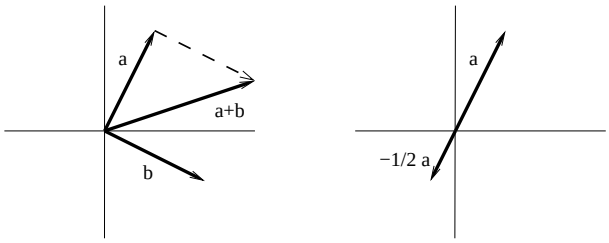

Figure 1: Left: Vector addition. Vector **b** is added to vector **a**. Geometrically the sum can be obtained by aligning the tail of **b** to the head of **a**. Right: multiplication of a vector with a scalar (-1/2).

clidean length of a vector is given by $|\mathbf{x}| = \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2 + \ldots}$. The distance between two vector is the length of the difference vector. Using the Euclidean norm the distance d is $d(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots}$. Distances have the following properties:

1. $|\mathbf{x} - \mathbf{y}| \geq 0$, and if $|\mathbf{x} - \mathbf{y}| = 0$ then $\mathbf{x} = \mathbf{y}$
2. $|\mathbf{x} + \mathbf{y}| \leq |\mathbf{x}| + |\mathbf{y}|$ (triangle inequality)
3. $|\alpha\mathbf{x}| = |\alpha||\mathbf{x}|$

A more general norm is the p-norm (with $p \geq 1$), it is defined as

$$|\mathbf{x}|_p = (|x_1|^p + |x_2|^p + |x_3|^p + \ldots)^{1/p}$$

The case $p = 2$ corresponds to the Euclidean norm, just described. When $p = 1$ one obtains the so-called Manhattan distance, $|\mathbf{x}|_1 = |x_1| + |x_2| + |x_3| + \ldots$. In a 2-dimensional plane it describes the distance between two points you would travel in Manhattan around the blocks. For instance, the distance between $\mathbf{p} = (42^{nd}str, 5^{nd}Av)$ and $\mathbf{q} = (38^{th}str, 2^{nd}Av)$ is $|p_1 - q_1| + |p_2 - q_2| = 7$ blocks.

One can also take the limit $p \to \infty$ and one has

$$|\mathbf{x}|_\infty = \max_i |x_i|$$

Sometimes this is called the chessboard distance. Check for yourself why the *max* appears, by taking a numerical example for a large value of p.

Finally, to calculate the distance between two binary vectors one has defined the Hamming distance. It is simply the sum of the mismatched bits. Note that when the type of norm is not explicitly mentioned, the Euclidean one is implied.

Unit vectors There are certain instances where we are more interested in the direction of the vector as than in its length. Normalised, or unit vectors have length one. They are indicated with a caret. We can easily create a normalized version of a vector by dividing it by its length.

$$\hat{\mathbf{x}} = \mathbf{x}/|\mathbf{x}|$$

Of course, $|\hat{\mathbf{x}}| = 1$.

Relevance to cognition

Distance is an important concept in perception. Because vectors can represent more abstract constructs such as visual stimuli (above), we can also talk about the distance between stimuli. Suppose we present two almost identical stimuli, we can wonder when a subject will perceive them as different, and what distance measure perception uses. You can already look at Fig. 2, to examine the different distances. There is no straightforward answer to these questions.

Another important application of distance measures is that they can be use to assess the quality of an artificial system. A distance can be used to measure the difference between the actual output of the system and the desired response. Because of the first property of the distance, zero distance would mean perfect performance. We will use this below.

1.3 Cluster plots

Suppose we have a bunch of data points in a high dimensional space, in other words a set of vectors, and we are interested in their structure and relation. How can we show the structure of the data and the relation between the data points? One

solution would be to simply display the data point, for instance, by representing each element of the vector as a grey scale. However this becomes tedious when there are many data points.

A cluster-plot (dendrogram) can be used to express pair-wise structure in the data, Fig 2. The algorithm to create a dendrogram is as follows: First, calculate all Euclidean distances between all possible pairs. Next, find the pair with shortest distance; connect the points with a fork with a length equal to the distance; replace the pair with their average; repeat this procedure until all data are connected.

As the example shows, data on the same branch of the tree are most similar.

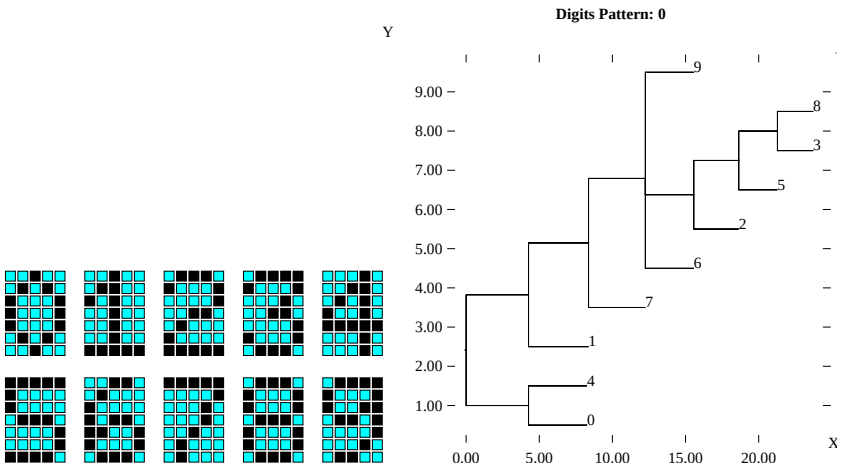

Figure 2: Left: 10 binary digit patterns which we will use for object recognition. Right: Cluster plot showing the distances between the digits.

1.4 Inner product

The inner product is also called dot product or scalar product. It takes two vectors and produces a single number. It is defined as

$$\begin{aligned} \mathbf{a}.\mathbf{b} &= \sum_{i=1}^{N} a_i b_i \\ &= a_1 b_1 + a_2 b_2 + \ldots \end{aligned}$$

The inner product can also be written as

$$\mathbf{a}.\mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\phi$$

here ϕ is the angle between the two vectors. Note that independent of the dimension of the space, the angle between two vectors is always properly defined.
Example: if $\mathbf{a} = (2, 0)$, $\mathbf{b} = (1, 1)$, then $\mathbf{a}.\mathbf{b} = 2$, which equals $2\sqrt{2}\cos\frac{\pi}{4}$

The inner product has the following properties:

- $\mathbf{a}.\mathbf{b} = \mathbf{b}.\mathbf{a}$
- $|\mathbf{a}.\mathbf{b}| \leq |\mathbf{a}||\mathbf{b}|$. This is known as the Schwarz-inequality. Note, that this follows from the fact that $|\cos\phi| \leq 1$.
- When the vectors are perpendicular their inner product is zero. Another way of saying this is that the vectors are ORTHOGONAL.

- The inner-product is related to the length of the projection of one vector on the other. When one vector, say **a**, is taken a unit vector, the inner product gives the length of the projection of **b** onto **a**. If **a** is not a unit vector, the inner-product is multiplied with **a**'s length.
- Two *unit* vectors whose inner product is one are necessarily parallel, and when the inner product is minus one, they are opposite.

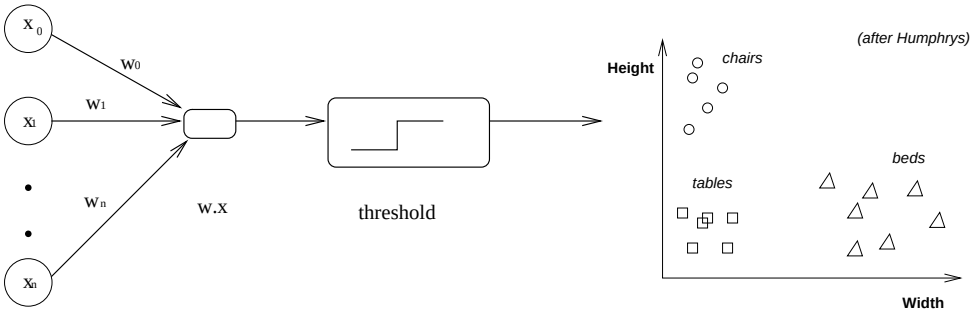

Figure 3: Left: a simple model of a neuron. The inputs x are weighted with w and summed, the sum is thresholded. Right: Furniture can be classified according to height and width.

Relevance to cognition

A neuron collects inputs from many different other neurons, typically some 1000 to 100000. Each connection, termed SYNAPSE in biology, has a certain strength, indicated by weight w_i. The set of all synapses is written as the vector $\mathbf{w}$. In the most basic models each w_i can take any value, positive or negative. When a weight is positive it is called excitatory, negative weights are called inhibitory.
Suppose that the activities of the input neurons are written as $\mathbf{x}$. A simple model of a neuron, common in cognitive modelling, is to say that the neuron's activity, or firing rate, r is

$$r = f(w_1x_1 + w_2x_2 + w_3x_3 + \ldots) = f(\mathbf{w}.\mathbf{x})$$

Where f is some function that translates the *net* input, $\mathbf{w}.\mathbf{x}$, to output activity. It is almost always a monotonically increasing function. There are a few common choices for f: the hyperbolic tangent, the logistic function, and the binary threshold

$$\begin{aligned} f(x) &= \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}} \quad \text{(the hyperbolic tangent)} \\ f(x) &= \frac{1}{1 + \exp(-\beta x)} \quad \text{(the logistic function)} \\ f(x) &= 0 \quad \text{if } x < 0 \\ &\quad\ 1 \quad \text{if } x \geq 0 \quad \text{(threshold function)} \end{aligned}$$

The situation is sketched in Fig. 3 left, for a thresholding unit. The parameter β gives the steepness of the transition. An additional parameter, the threshold, can also be included. For example, $f(x) = \tanh(\beta(x - T))$, where T denotes the threshold. Its role is to shift the activation curve.

From the above we know that the input is projected onto the weight vector. This means that the most effective stimulus vector will be the one which

aligns with the weight vector. The neuron acts as a template matcher, where the template is given by the weight vector.

- Despite their simplicity these nodes are very powerful. McCullough and Pitts have shown some 60 years ago that networks of these nodes can do every kind on binary computation. So there is no limit to the complexity even when we stick with these simple nodes.

- One of the main assumptions in neuroscience is that the weights store memory, and their are many ideas how the weights can be changed in order to change the memories stored or the computation that the network performs (see below).

- It is important to realise that this model assumes linear summation of the inputs, namely the net input is $\mathbf{w}.\mathbf{x}$. This is convenient but this has not been proved convincingly in real neurons. In some cases it is known that the inputs interact locally.

Defining planes with the inner-product The inner product also can be used to define a plane. In n dimensions the equation $\mathbf{x}.\mathbf{a} = c$, where $\mathbf{a}$ is a given vector and c is a given constant defines a $(n-1)$ dimensional plane for the vector $\mathbf{x}$. (planes in arbitrary dimension are also called hyper-planes). In two dimensions this simplifies to $a_1x_1 + a_2x_2 = c$ which defines a line.

Relevance to cognition

Categorisation is the problem of labelling inputs with the right category label. In the case of visual recognition we might want to label a picture as containing a car or not. We categorize all the time, both consciously and unconsciously. In simple cases the categorisation can be done by determining on which side of the hyper-plane the data falls. Suppose the input is given by $\mathbf{x}$, one can now calculate $\mathbf{x}.\mathbf{a}$ with $\mathbf{a}$ some smartly chosen vector and compare the outcome to a smartly chosen constant c. We can construct a detector which when the outcome is smaller concludes that the input was a chair, when larger, the input was a table or a bed, see Fig. 3.

1.5 Basis

A set of vectors forms a basis when each vector in the space can be uniquely described by a linear combination of these basis vectors.

The most common basis is the orthonormal basis, which in three dimension are the x, y and z direction. The basis vectors are commonly written as $\mathbf{e_1} = (1,0,0)$, $\mathbf{e_2} = (0,1,0)$, and $\mathbf{e_3} = (0,0,1)$. The vector $\mathbf{a} = (2,3,0)$ can be decomposed as $\mathbf{a} = 2\mathbf{e_1} + 3\mathbf{e}_2 + 0\mathbf{e_3}$.

A basis has as many basis-vectors as there are dimensions in the space. But not every set of vectors forms a basis. When the vectors are LINEARLY DEPENDENT they do not form a basis. For example, $(2,0,0)$, $(4,2,0)$, and $(0,1,0)$ do not form a basis because they are linearly dependent (see exercises).

For an ORTHONORMAL basis one has for the inner products $\mathbf{e}_i.\mathbf{e}_j = \delta_{ij}$ where δ_{ij} is the Kronecker-delta, $\delta_{ii} = 1$, and $\delta_{ij} = 0$ if $i \neq j$.

For an ORTHOGONAL basis the basis vectors are orthogonal ($\mathbf{e}_i.\mathbf{e}_j = 0$ if $i \neq j$), but not normalized ($\mathbf{e}_i.\mathbf{e}_i = c_i$).

1.5.1 Matlab notes

Matlab (or its free clone 'octave') is ideal to work with vectors and matrices. You can use the function 'dot' to calculate dot products. The function 'norm' calculates

the norm of the vector. We first define two vectors

```
octave:1> a=[1 2 1]% size=1x3, row vector
a =1 2 1
octave:2> b=[1 1 0]' % size=3x1, column vector
b =1
1
0
octave:3> dot(b,a) % calculate the dot product
ans = 3
octave:4> norm(a) % default euclidean norm
ans = 2.4495
octave:26> norm(a,1) % the Manhattan distance
ans = 4
```

Note, that we have defined two slightly different vectors, a row vector $\mathbf{a}$ and a column vector $\mathbf{b}$, and the actual screen output will reflect this. This distinction will be important in the next section. Usually vectors are column vectors by default, and $\mathbf{a}^T$ can be used to indicate a row vector.

1.6 Exercises

1. Calculate $(1, 2, 3) - (-1, 2x - 2, x)$. What does this set of vectors describe when x is a variable.

2. We have discussed that the retinal input can be interpreted as a vector of high dimensionality. What do scalar multiplication and addition correspond to in the context of retinal inputs?

3. For the 3 different distance measures check that they obey the criteria, mentioned in 1.2.

4. Calculate the distance between $(1, 0, 2)$ and $(3, 1, -1)$ for the 3 different distant measures.

5. If we change a single pixel in an image, will we perceive an image as different? What if we change the brightness of the picture? What is the distance between the original and modified image in each case. Discuss your results.

6. Calculate $\mathbf{a}(\mathbf{b}.\mathbf{c})$ with $\mathbf{a} = (1, 0)$, $\mathbf{b} = (2, 1)$ and $\mathbf{c} = (2, 3)$.

7. Given the vector $(3, 4)$, create a vector that has the same direction but with length 2.

8. Show that $|\mathbf{a} - \mathbf{b}|^2 = |\mathbf{a}|^2 + |\mathbf{b}|^2 - 2\mathbf{a}.\mathbf{b}$. Hint: write out the products.

9. How would the data from Fig. 3right, look in a cluster plot ?

10. Plot the activation function $f(x) = \tanh(\beta(x - T))$ against x for a few choices of β and threshold T.

11. Given the line $\mathbf{a}.\mathbf{x} = c$ in two dimensions, how would you write a computer program that draws it?

12. Given vectors $(1, 2, 1)$, $(3, 1, 0)$, $(2, -1, -1)$. Check that some points, such as $(1, 0, 0)$ can not be described as a sum of these vectors, on the other hand there are many possible decompositions of the vector $(4, 3, 1)$. Do the three vectors form a basis?

2 Matrices

Matrices are convenient representations of linear operations on vectors. They consist of rows and columns. An $m \times n$ matrix has m rows and n columns. So a 3x4 matrix looks like

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}$$

Individual elements are indicated again with subscripts. Importantly the matrix performs a linear operation on a vector. When the matrix $(m \times n)$ operates on a n-dimensional vector, the output is a vector (m-dimensional). The matrix operation on a vector is written as $\mathbf{w} = A.\mathbf{v}$ or $\mathbf{w} = A\mathbf{v}$. It is defined as

$$w_i = \sum_{j=1}^{n} a_{ij} v_j$$

For example, if $A = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix}$ and $\mathbf{v} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$, $A\mathbf{v} = \begin{pmatrix} 1.1 + 2.3 \\ 3.1 + 0 \end{pmatrix} = \begin{pmatrix} 7 \\ 3 \end{pmatrix}$.

We can subtract and add matrices in a component-wise fashion, $C = A + B$, means that $C_{ij} = A_{ij} + B_{ij}$. Of course the matrices A and B need to be the same size for this operation. Scalar multiplication is also component wise, if $C = \alpha A$, then $C_{ij} = \alpha A_{ij}$. Note, that these definitions are much like the definitions for vectors.

Matrix multiplication

Matrices can also be multiplied with each other to form a new matrix, this way $k \times m$ matrix A, is multiplied with $m \times n$ matrix B. The result is again a matrix, a $k \times n$ matrix, it is written as $C = A.B$, the components of C are

$$C_{ik} = (A.B)_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk}$$

The new matrix transforms a n dimensional vector into a k dimensional one. It does not matter in which sequence we do the multiplication, that is $A(B\mathbf{v}) = (AB)\mathbf{v}$, and also $A(BC) = (AB)C$. Applying a number of matrix multiplications subsequently on a vector, is identical to applying the matrix product on the vector. We can also define powers of matrices such as $A^3 = A.A.A$ using the above multiplication rule.

Be extremely careful: If we have two numbers x and y, then $xy = yx$. But the matrix products are not necessarily *commuting*, that is, in general $A.B \neq B.A$. For instance $\begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 3 \end{pmatrix}$, while $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 0 & 0 \end{pmatrix}$. With the matrices in the next section, you should be able to create examples of both commuting (when $AB = BA$) and non-commuting products (when $AB \neq BA$).

Linearity

Matrix operations are linear, because they obey the two REQUIREMENTS FOR LINEARITY:

1) $A(\alpha\mathbf{x}) = \alpha A\mathbf{x}$
2) $A(\mathbf{x} + \mathbf{y}) = A\mathbf{x} + A\mathbf{y}$

You can easily prove this using the definitions. Linearity is a very important property. If a system is linear and once we know how a few vectors transform, it allows us to 'predict' the outcomes of a certain matrix transformation on any given vector.

Relevance to cognition

> Matrices can be used to describe the weights for one layer on neurons to the next layer. When we dealt with a single output neuron, the weight were a vector, but if we have multiple outputs, the weights can be conviniently written as a matrix.
> The product of linear transformations, such as $AB\mathbf{x}$, is again a linear transformation. That is nice for mathematicians, but a bit boring. When dealing with neural networks this means that adding extra layers to a network with linear neurons (stacking linear operations) will not increase its computational power. This changes dramatically when the nodes have a non-linear input-output relation, like the examples in the previous chapter and the perceptron of the next chapter. Having a non-linear neuron hugely increases the number of possible transformations, but the convenient results from linear algebra are no longer valid.

Transpose of a matrix

Transpose denoted A^T denotes the transpose of the matrix. It is the matrix with the rows and columns interchanged, so $(A^T)_{ij} = A_{ji}$. Example in two dimensions $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $A^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$. We call a matrix symmetric if $A^T = A$.

As an aside, mainly mentioned for completeness: when dealing with complex-valued matrices a complex conjugation is often combined with transposing, this is called matrix conjugation. Example $A = \begin{pmatrix} a+bi & c+di \\ e & f \end{pmatrix}$, $A^\dagger = \begin{pmatrix} a-bi & e \\ c-di & f \end{pmatrix}$

2.1 Common matrices

Some common linear transformations in two-dimensions are rotation, mirroring and projection.
Rotation over an angle ϕ is given by the matrix

$$R_\phi = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

For example $R = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ rotates vector over 90^o.
Mirroring in a line which has angle ϕ w.r.t to the x-axis:

$$M_\phi = \begin{pmatrix} \cos\phi & \sin\phi \\ \sin\phi & -\cos\phi \end{pmatrix}$$

Projection onto the vector $\mathbf{a} = (a_1, a_2)$, with $|\mathbf{a}| = 1$, can be done with a matrix

$$P = \begin{pmatrix} a_1^2 & a_1 a_2 \\ a_1 a_2 & a_2^2 \end{pmatrix}$$

For any projection matrix one has $P^2 = P$. This makes sense as projecting an already projected vector, $P(P.\mathbf{v})$, should not change it anymore. It is a good exercise to check for the above matrix. Note, the projection matrix operating on a vector gives the projected vector, whereas the inner product calculates the *length* of the projection.

2.2 Determinants

The determinant of the matrix is a useful concept. One way to introduce it, is the following: Suppose we create a unit (hyper)-cube with the basis vectors. For instance, in two dimensions the basis vectors $(0,1)$ and $(1,0)$ span the square $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$. Now we transform the vectors that span the square. The square is mapped to a parallelogram (or parallelepiped in 3D). Now measure its area spanned by the transformed vectors. The ratio in the area turns out to be a useful quantity and is given by the absolute value of the determinant. The determinant is also denoted $|A|$.

For a 2×2 matrix the determinant is

$$|A| = det(A) = det \begin{pmatrix} a & b \\ c & d \end{pmatrix} \equiv ad - bc$$

For example, the determinant of the rotation matrix is $\det(R) = \cos^2\phi + \sin^2\phi = 1$, which shoudl not surprise you given our discussion of the interpretation of the determinant.
For a 3×3 dimensional matrix

$$det(A) = det \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \equiv aei + bfg + cdh - gec - hfa - idb$$

The equation in higher dimensions is more tedious, but Matlab won't have a problem with it. The function is called `det(A)`.

Unlike the ratio of areas, the determinant can also be negative. The sign is negative when the parallelogram is flipped by the matrix transformation.

The determinant can also be used to check whether a certain set of vectors span a basis. Hereto line up the column vectors to create a matrix. If the determinant is zero, the vectors are linearly dependent and hence do not form a basis.

2.3 Identity and inversion

The simplest transformation leaves the vectors unaffected. This transformation is given by the identity matrix denoted I. It has zeros everywhere except on the diagonal, so in two dimensions $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. You should check that for any vector $\mathbf{v}$, one has $\mathbf{v} = I.\mathbf{v}$. The identity matrix has only elements on the diagonal, such matrices are called *diagonal* matrices, they have simplified properties: the determinant is simply the product of all diagonal elements, and all diagonal matrices commute with each other.

Most square matrix transformations have an inverse, denoted A^{-1}. Inverses only exists for square matrices (although more general concepts such as pseudo-inverses have been defined for non-square matrices). The inverse is implicitly defined as

$$A.A^{-1} = I$$

It basically says that the inverse should undo the matrix transformation. If A is a $n \times n$ matrix then this equation actually consists of one equation per element, hence we have n^2 equations. We have n^2 unknowns, because the unknown matrix A^{-1} has n^2 entries. For instance, suppose $A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ if we write the elements of A^{-1} as b_{ij} we have $b_{11} + 2b_{21} = 1$, $b_{12} + 2b_{22} = 0$, $b_{21} = 0$, $b_{22} = 1$. These equations are independent and we can solve this, and thus find A^{-1}. There are more sophisticated methods to calculate inverses. Matlab command is `inv(A)`.

But not every matrix has an inverse. A necessary and sufficient condition is that the determinant of the matrix is non-zero. A counter-example is given by the projection matrix. Its determinant is zero, and indeed, one cannot invert or undo the projection as after a projection the original vector is unknown.

Note that the identity matrix is its own inverse.

When calculating with matrices and inverses it is important to keep in mind that they do not always commute. Therefor we have to keep track of the order of the terms. For instance, if we have $A^{-1}B = C$, we can multiply from the left with A to get $AA^{-1}B = AC$ or $B = AC$. But a multiplication from the right gives $A^{-1}BA = CA$.

2.4 Solving matrix equations

If we have a set of linear equations, we can use matrices to solve them. Suppose we want to solve $x_1 + 2x_2 = 5$, $3x_1 = 2$ for x_1 and x_2 (the typical "John is five years older then Mary, and twice her age" type of problem). This is conveniently rewritten as a matrix equation $A\mathbf{x} = \mathbf{y}$, where $A = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix}$, $\mathbf{y} = (5, 2)$.

Suppose we want to solve an equation $A\mathbf{x} = \mathbf{y}$, where A and $\mathbf{y}$ are given. There are a few scenarios:

- This equation will have an unique solution when $\det(A) \neq 0$. The solution is simply $\mathbf{x} = A^{-1}\mathbf{y}$.
- In the special case that $\det(A) \neq 0$ and $\mathbf{y} = \mathbf{0}$ the only solution is $\mathbf{x} = \mathbf{0}$.
- When $\det(A) = 0$ there is a whole hyper-plane of solutions when $\mathbf{y} = \mathbf{0}$.
- Finally, when $\det(A) = 0$ but $\mathbf{y} \neq 0$, there can be many or no solutions.

2.5 Eigenvectors

Eigenvectors are those vectors that maintain their direction after the matrix multiplication. That is, they obey

$$A\mathbf{v_i} = \lambda_i \mathbf{v_i} \tag{1}$$

where λ_i is called the (i-th) EIGENVALUE, and $\mathbf{v_i}$ is called the corresponding EIGENVECTOR (there is no standard letter for the eigenvector). There are as many eigenvectors as there are dimensions.

To find the eigenvalues we write the above equation as $(A - \lambda I)\mathbf{v_i} = 0$, next we look for $\mathbf{v}_i$which solves this. The equation has always trivial solutions, $\mathbf{v_i} = \mathbf{0}$, but those don't interest us. From above we know that for this equation to have a non-trivial solution, we need $\det(A - \lambda I) = 0$. The eigenvalues are those values of λ that satisfy this equation. To find them we write out the definition of the determinant, this will yield a polynomial in λ with an order equal to the number of dimensions. The solutions to this polynomial are the eigenvalues.

Next, we need to determine the eigenvectors. Each eigenvector has an associated eigenvector. To find the eigenvectors, one plugs in a found eigenvalue and solves Eq.(1).

Example for the mirroring matrix: $\det(A-\lambda I) = \det \begin{pmatrix} \cos\phi - \lambda & \sin\phi \\ \sin\phi & -\cos\phi - \lambda \end{pmatrix} =$ $\lambda^2 - 1$. So in order for the determinant to be zero, we need $\lambda = \pm 1$. So the eigenvalues are 1 and -1. Next, we calculate the eigenvector associated to $\lambda = 1$. If we write this eigenvector as $\mathbf{v} = (x_1, x_2)$, we have $A.\mathbf{v} = \lambda\mathbf{v}$, or

$$\begin{aligned} x_1 \cos\phi + x_2 \sin\phi &= x_1 \\ x_1 \sin\phi - x_2 \cos\phi &= x_2 \end{aligned}$$

One solution to these equations is $(x_1, x_2) = (1 + \cos\phi, \sin\phi)$. But it is not difficult to see that there is a whole line of solutions, as the eigenvector's length is not fixed.

- Eigenvalues can be zero.
- Some eigenvalues can share the same value, this is called degenerate.
- When the eigenvalues are not degenerate, the eigenvectors form a basis.
- Eigenvalues can also be complex, as is the case for the rotation matrix.
- When the matrix is symmetric, the eigenvalues are real, and what is more, the eigenvectors form an orthogonal basis.[1]

Using eigenvectors can be very convenient in particular when they form a basis. Because rather than having to study a full matrix transformation now we only need to describe the transformations of the eigen vectors. In other words, the equations de-couple. We will see some examples below.

2.6 Covariance matrix

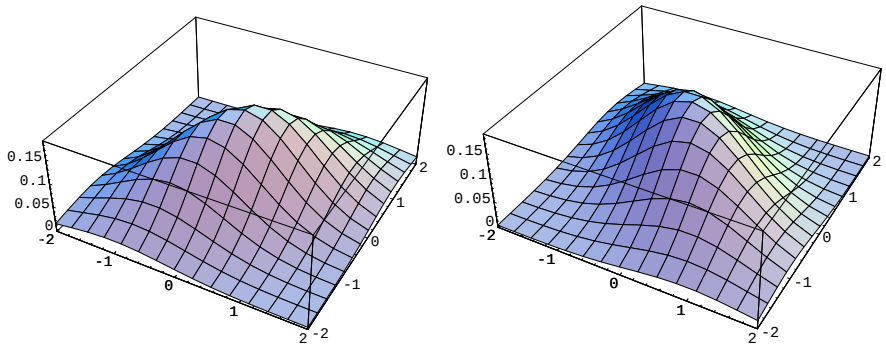

Figure 4: Two dimensional Gaussian distributions. Left: the correlation matrix was $C = \begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}$ and Right: $C = \begin{pmatrix} 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}$ (anti-correlated). The mean was $(0, 0)$.

A special matrix is the covariance matrix. It describes how observed data points are related. It is therefore important in data-analysis and statistics. Before we can describe it, we need some statistics.

Given a probability distribution $P(x)$, which is normalized such that $\int dx\, P(x) = 1$. Now the mean is

$$\mu = \langle x \rangle = \int dx\, P(x) x$$

The triangular brackets denote the average. The variance is given by[2]

$$\sigma^2 = \langle \delta x^2 \rangle = -\langle x \rangle^2 + \int dx\, P(x) x^2$$

[1] In case of matrices with complex entries: if the matrix is Hermitian: $A^\dagger = A$, which means $A^*_{ij} = A_{ji}$, then 1) all eigenvalues are real, and 2) the eigenvectors form a orthogonal basis of the space.

[2] Similar definitions hold when we measure the statistics of given data. This is called DESCRIPTIVE STATISTICS. Suppose we measure N instances of a data x^μ. The mean vector is defined as $\langle x \rangle = \frac{1}{N} \sum_\mu x^\mu$; the variance is given by $\langle \delta x^2 \rangle = \frac{1}{N-1} \sum (x^\mu - \langle x \rangle)^2 = \frac{1}{N-1} \sum_\mu (x^\mu)^2 - \langle x \rangle^2$.

You have probably already encountered the one-dimensional Gaussian distribution

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp[-(x-\mu)^2/2\sigma^2]$$

This distribution has mean μ, variance σ^2, and like any decent probability distribution it is normalized. You can check these three facts, but these are tricky integrals; you could try it with 'maple' or 'mathematica'.

When we deal with higher dimensional distributions, similar definitions hold. Note, the mean becomes a vector. The variance is slightly more tricky. The variance generalizes to a matrix, the covariance matrix. The covariance between two components of the vector is defined as

$$C_{ij} = \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle$$

The covariance matrix C has entries C_{ij}. Note that by construction the matrix is symmetric.

When the components of $\mathbf{x}$ are independent the matrix has only diagonal terms, as all terms for which $i \neq j$ disappear.

A multi-dimensional Gaussian distribution is given by

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det(C)}} \exp[-\frac{1}{2}(\mathbf{x}-\mu)C^{-1}(\mathbf{x}-\mu)] \tag{2}$$

When the components of $\mathbf{x}$ are independent, the matrix C and its inverse are diagonal. As a result the distribution factorizes as expected as $P(\mathbf{x}) = P(x_1)P(x_2)\ldots P(x_N)$.

When the two variable are correlated, the probability distribution looks as given in Fig. 4. For instance, in the left case, the variables are positively correlated. It is more common to find a larger value for x_2 when x_1 is large. In other words, x_1 *predicts* x_2.

However, always be careful to not confuse correlation with causality. Saying that poor education and a shorter life-expectancy are correlated, does not mean one causes the other...

Relevance to cognition

> We have already seen that the size of the visual input space is tremendous. However, there are often correlations present between the inputs. One believes that the nervous system is tuned to deal with these correlations and extract information from them. (see exercises)

2.6.1 Matlab notes

In Matlab the '*' operator implements a matrix product.

```
octave:1> m=[1,2;3 ,-1]
m =
1 2
3 -1
octave:2> m*[2 0]'
ans=
2
6
```

In the previous chapter we used the `dot` command to calculate inner products. You can also write inner products using the '*' operator. Note, that the definition of the inner product and the matrix product are quite similar. To Matlab vectors are nothing but $1 \times n$ or $n \times 1$ matrices. The apostrophe introduced earlier transposes

the matrix, for vectors this means that a column vector becomes a row vector and vice versa. When you use '*', you have to be careful whether the vectors are column or row vectors.

```
octave:1> a=[1 2 1]% size=1x3
a =1 2 1
octave:2> b=[1 1 0]' % size=3x1
b =1
1
0
octave:15> a*b % inner product
ans = 3
octave:5> b*a % 'tensor' or 'outer' product
ans =
1 2 1
1 2 1
0 0 0
```

Matlab's convention is that it sums over the inner indexes. As you see multiplying a 1×3 with a 3×1 array gives a 1×1 array, i.e. a scalar (the middle indexes are contracted). On the other hand multiplying a 3×1 with a 1×3 array gives a 3×3 array, or matrix. If the dimensions don't match and you try to multiply, it will complain.

```
   octave:6> m*a %
m*a error:  operator *:  nonconformant arguments (op1 is 2x2, op2 is 1x3)
```

Matrix functions in Matlab Matlab has many special matrices and matrix functions

```
   octave:6> det(m) % determinant
ans= -7
   octave:7> lam=eig(m) %eigen values of m
lam =
2.6458
-2.6458

   octave:7> [v,lam]=eig(m) % eigenvalues and eigenvectors of m
v =
0.77218 -0.48097
0.63541 0.87674

   lam =
2.64575 0.00000
0.00000 -2.64575

   octave:9> eye(3) %identity matrix in 3 dimensions
ans =
1 0 0
0 1 0
0 0 1
```

Unless you tell Matlab otherwise, all multiplications are matrix multiplications. However, in some cases you want to multiply vectors or matrices component-wise. For instance, the components of **a** describe the unit-price of each item in a list of items, and **b** describes the quantity of each item, then the total price for the different items is $c_i = a_i b_i$. In Matlab we can calculated this using the '.*' operator.

```
   octave:10> a=[0.12 0.4]; b=[2 3]; a.*b
```

```
ans= [0.24 1.2]
```

Even more tricky is the division operator. With `a/m` will solve $\mathbf{c} = \mathbf{b}A^{-1}$, that is the $\mathbf{c}$ for which $\mathbf{c}A = \mathbf{b}$. (In some applications this is a useful thing to do, however, for us it is mainly confusing).

```
   octave:10> c=a/m
c= [0.188 -0.022]
octave:11> c*m % check the solution.
ans= [0.12 0.4]
```

Using the command `a/b` when `a` and `b` are vectors, Matlab will *try* to solve c=b/a. As this is an ill-formed matrix equation, and Matlab will return the least square solution. To do component-wise division use the `'./'` operator. Finally, also the power operator ^ has the variant '.^'.

2.7 Exercises

1. Calculate the matrix product AB, where $A = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$ and $B = \begin{pmatrix} -2 & 0 \\ 3 & 2 \end{pmatrix}$. Also calculate the determinants of the individual matrices and of the product matrix. Given the geometric definition of the determinant, do you think that in general $\det(AB) = \det(A)\det(B)$?

2. The product of linear transformations, such as $A(B\mathbf{x})$, is again a linear transformation. Show this.

3. The translation that maps $\mathbf{v}$ to $\mathbf{w}$ is given by $\mathbf{w} = \mathbf{v}+\mathbf{c}$, where $\mathbf{v}$ is the original vector and $\mathbf{c}$ is some constant vector. This is also a common transformation, but it is not linear. Show why.

4. Suppose you have an unknown 3x3 matrix, but you can apply it to any vector you want and read out the transformed vector. What is the easiest way to find out all the matrix entries?

5. As mentioned, the matrix product do not always commute. Take a rotation over $\pi/2$ followed by a mirroring in a line that has zero angle with the x-axis. What is the matrix that describes the product of these two operations, i.e. calculate $M_0 R_{\pi/2}$. Now also calculate $R_{\pi/2} M_0$. Is the answer the same? Draw a picture how a vector like $(1, 1)$ is transformed under both transformations and explain. Without explicit calculation determine whether $R_\pi R_{\pi/2} = R_{\pi/2} R_\pi$.

6. Calculate the determinants of the rotation, mirroring and projection matrices. Interpret your result.

7. What are the inverses of the mirroring and rotation matrices? And of the projection matrix? Either calculate or think.

8. Solve, if possible, each of the following sets of equations: $\{x+y = 1,\ x-y = 2\}$, $\{x+y = 0,\ x+y = 2\}$, $\{x+y = 0,\ x-y = 0\}$, $\{x+y = 0,\ x+y = 0\}$. Check with Section 2.4.

9. Check that $(x_1, x_2) = (1+\cos\phi,\ \sin\phi)$ is indeed an eigenvector of the mirroring matrix. What is the other one? Sketch them together with the line with angle ϕ.

10. What is the determinant of a diagonal matrix in two or three dimensions? What about n dimensions?

11. Like Fig. 4, plot in a contour-plot the joint Gaussian distribution in case $C = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$. (Note how easily this matrix is inverted). Using the definition of the multi-dimensional Gaussian distribution Eq. 2, check that the distribution factorizes.

12. Generating a pair of uncorrelated random Gaussian distributed variables is easy on a computer (e.g. '`randn(1,2)`' in Matlab). How can you generate correlated or anti-correlated numbers?

13. Consider an arbitrary photograph of an object. Now pick an arbitrary pixel in it. Which pixels are strongly correlated with the chosen pixel?

3 Application: Perceptron

3.1 More about neurons

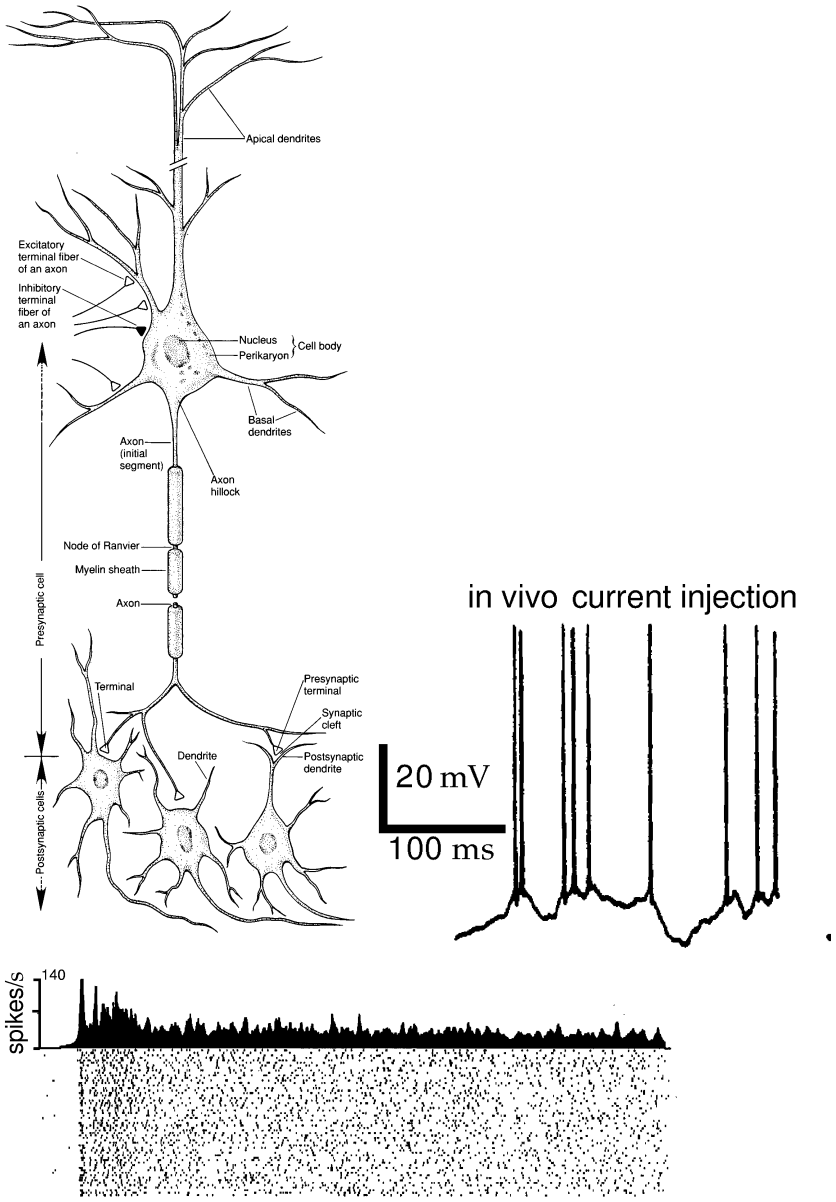

Figure 5: Left: Cartoon of a biological neuron. Middle: Real neurons produce action potentials or spikes in response to excitatory input.
Right: In this case activity from a motion detection neuron was measured over some 50 trials. In every trial the spikes came at different times (bottom raster plot, each dot is a spike). The averaged activity is given by the firing rate (top plot), typically ranging from 0 to a few hundred Hz.

The human brain contains about 10^{11} neurons, each with about 10^4 inputs (synapses). Biological neurons produce stereotypic discrete events, called action potentials or spikes in response to excitatory input, Fig. 5.

In the following we will describe the neurons with their firing rate rather than their precise spike times. One argument is that the timing of the spikes in biology is often not very precise and varies from trial to trial, Fig. 5right, so maybe precise timing does not matter. Whether this is always a good approximation to describe the biological brain is not very clear.

A more relaxed attitude that we will follow here is that the biological systems are an inspiration for our thinking, and we are simply interested in what neuron-like computations can do.

3.2 Perceptron

The perceptron is one of the simplest neurally inspired supervised learning systems. It learns to do categorisations. In supervised learning a teacher is present that judges the result, usually after each trial, and feeds back so that the network can be adjusted. [In contrast, in unsupervised learning the learning rule extracts information about world by itself by using statistical properties of the world.] The perceptron is shown in Fig. 6.

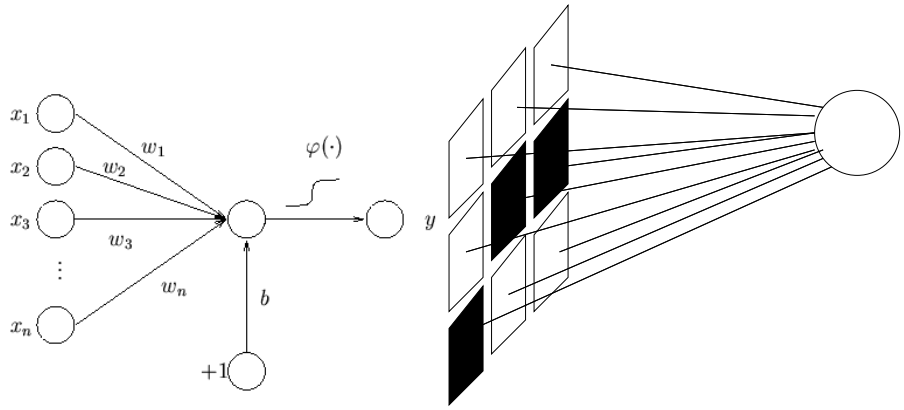

Figure 6: The perceptron: n inputs are summed with weights w to give the total input, which is goes into a non-linearity. The additional input is the bias b can be used to set the threshold of the unit.
Right: Connecting the perceptron to a retina.

For convenience we will assume the inputs are on some kind of retina, Fig. 6. One typical task is to classify images. For instance, can we construct a digit detector which is robust to noise? We write the inputs as a vector $\mathbf{x}$, and write weights as vector $\mathbf{w}$. The total input is now $\mathbf{x}.\mathbf{w}$. (Both $\mathbf{w}$ and $\mathbf{x}$ are unconstrained which is slightly un-biological). We label different patterns with index μ. Pattern 1: $\mathbf{x}^{\mu=1}$ etc. Although in the end it would be nice to have multiple output units, one for each digit (see Fig. 7left), we start with a single output node.

Suppose we have a single binary node, where the output y is given by

$$y = 2H(\mathbf{w}.\mathbf{x}) - 1$$

Where $H(x)$ is the Heaviside step function: $H(x) = 0$ if $x < 0$, $H(x) = 1$ if $x > 0$. That means that the output is either +1 or -1, depending on whether $\mathbf{x}.\mathbf{w}$ was greater or less than zero. (this is a slight variant from above, but it is more practical here). The data set we want to classify consists of one subset of data that should yield a '-1' response, the other subset should yield '1' response $z^\mu = 1$. The z is the known desired outcome, in other words, the class labels. In our example, we can set $z = 1$ for '8'-like figures, and -1 for all other figures.

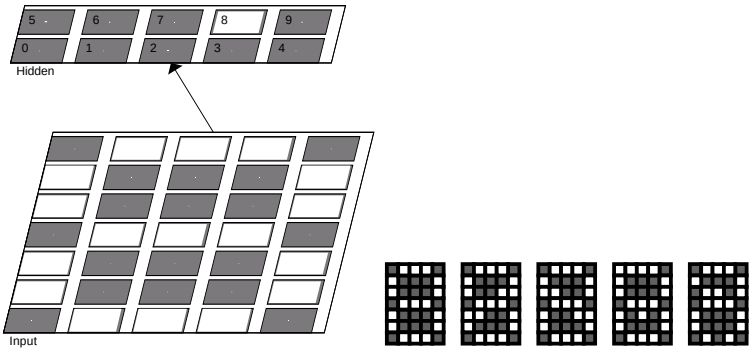

Figure 7: Left: An array of perceptrons (top layer), each responsible for a different digit.
Right: We would like the detector to be robust against small variations in the input (noise). So the depicted variants of the input should be detected as an '8' as well.

3.3 Perceptron learning rule

We now present examples from the set to the perceptron. For each example we determine if the perceptron produced the right response and if not, we adjust the weights. The perceptron can be trained as follows: Loop over the input data and apply the perceptron learning rule:

- If output was correct, don't do anything.
- Change the weight if output o was wrong:

$$\begin{aligned} \Delta w_j &= \eta(z^\mu - y^\mu)x_j^\mu \\ w_j &\Rightarrow w_j + \Delta w_j \end{aligned}$$

where η is a parameter called the learning rate which determines how much the weights are updated each time (its value is not important in this particular learning rule). The full perceptron learning rule is slightly more complicated leaves a little gap between the two categories (for details see [Hertz et al., 1991]). Using the full perceptron learning rule it can be proved that learning stops and if the classification can be learned, it converges in a finite number of steps.

An important question is whether the perceptron will work on any type of binary classification problem. It turns out that not every problem be learned by the perceptron. The perceptron can only learn linearly separable categorization, i.e. the data-points can be divided by a line or plane. Consider for instance the boolean functions of two variables. These can be represented in the two-dimensional plane, Fig. 8. The AND and OR function are separable, but the XOR (exclusive or) and the IDENTITY function are not linearly separable. This means these functions cannot be learned or represented by the perceptron, however, a layered network will do the trick, as we will see below.

Bias

So far we have assumed that separation plane goes through origin. This restriction can be avoided by having one additional 'always on' input and train its weight as well. For the AND you would for example present patterns (1,0,0), (1,0,1), (1,1,0), (1,1,1). This introduces a BIAS (a trick to remember !). The weight of the bias term is trained just with the above rule.

in	AND	OR	XOR	IDENTITY
-1,-1	-1	-1	-1	1
-1,1	-1	1	1	-1
1,-1	-1	1	1	-1
11	1	1	-1	1

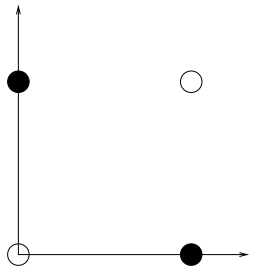

Figure 8: Left: Truth table of various Boolean functions. Right: Representation of the XOR function in the plane. The XOR is one only when either input is one, but not when both are one.

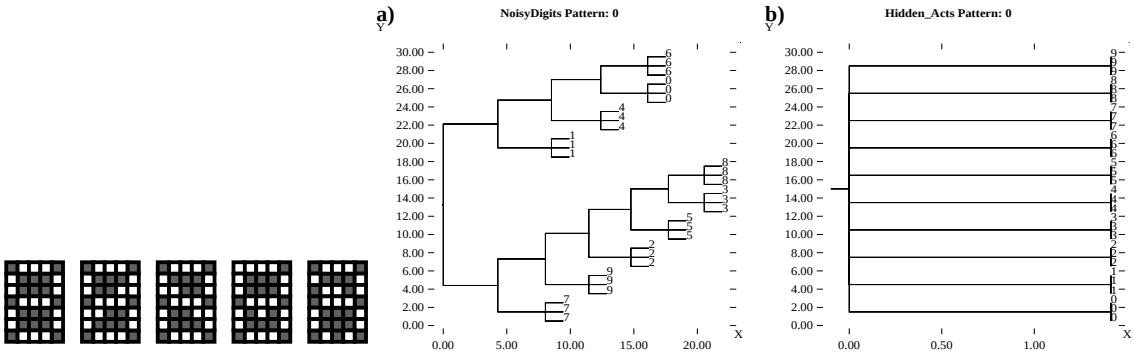

Figure 9: Transformation of the input patterns. The dendrogram of the input patterns (10 digits + distortion) is shown left and middle. Right: the dendrogram of the output activations for a network with ten outputs after learning.

Using perceptrons we can easily create a digit recognizer with multiple outputs, one for each digit. When a particular digit is presented, only the correct node becomes active. We can then test the network on a set of digit images and distorted images. The dendrogram of the inputs is shown on the left, the dendrogram of the output is on the right, Fig. 9. Note the generalization in right plot, all distorted versions of the '8' give an identical response. Such single bit errors are easily corrected by the network. However this digit recognizer is not really state of the art, and other errors and natural variations amongst the inputs can not be easily corrected. The network will fail badly if the digits are scaled in size or shifted along the retina.[3]

Relevance for -computation-

Note that computation and memory are tightly linked together in the perceptron. This is very different from a von Neumann computer where memory and processing are split. A von Neumann computer would take the input, retrieve the weight vector from memory, calculate the inner product in the central processing unit, and then go on to the next operation. Here processing is done in parallel, and can be, in principle, much faster, but as a disadvantage, we need a different node for each type of computation.

[3] An advanced digit recognizer using neural nets can be found at: `http://yann.lecun.com/`

3.4 Exercises

1. Sketch the AND function in the same way that the XOR was shown. Now construct a perceptron with the right weights that implements it. (Note, bias !)

2. Discuss the use and realism of the perceptron and its learning rule.

3. Perceptron algorithm. Apply the perceptron algorithm to between the following two sets of data vectors: set 1:$(1,0)$ and $(1,1)$ for which we want the perceptron to give a 1 as ouput; and set 2: $(-1,0)$ for which we want the perceptron to ouput -1. Cycle through all data until all points are classified correctly. Sketch the data and the final weight vector. Note that when you start with $w=(0,0)$ the output will be undefined; in order for the perceptron to be robust to noise, it is good to apply the learning rule in this case as well. Optional: Create a non-learn-able data-set, and see how the learning progresses.

4. Template matching and noisy inputs. As indicated in the lectures, a single neuron can be seen as a template matcher. Here we study template matching in the presence of noise. Suppose we have a two-dimensional input $\mathbf{x^a}$ and a background signal $\mathbf{x^b}$, both corrupted with independent Gaussian noise. No matter how well we train the perceptron, it will sometimes make errors because of the noise. Suppose the average signal to be detected is $\langle \mathbf{x}^a \rangle = (1,2)$ and the average background is $\left\langle \mathbf{x}^b \right\rangle = (0,0)$ (with $\langle . \rangle$ we denote an average). Assume first that the noise equally strong along both dimensions.
   a) How would you choose the weight-vector in this case so as to minimize the errors?

   Suppose now that the noise is stronger in one input than in the other one. The two-dimensional probability distribution $P(\mathbf{x^a}) = \frac{1}{\sqrt{2\pi}\sigma_1}\frac{1}{\sqrt{2\pi}\sigma_2}\exp[-(x_1^a - \langle x_1^a \rangle)^2/2\sigma_1^2]\exp[-(x_2^a - \langle x_2^a \rangle)^2/2\sigma_2^2]$, where $\mathbf{x^a} = (x_1^a, x_2^a)$ and σ_1 and σ_2 are the standard deviations in the two inputs. $\mathbf{x^b}$ has the same distribution.
   b) Indicate roughly how would choose the weight vector in the case that $\sigma_1 \ll \sigma_2$ and $\sigma_1 \gg \sigma_2$. The best weight vector causes the least overlap in the projections. Making a sketch of the situation is helpful. Draw in the x-plane the mean values of x^a and x^b and indicate the noise around them with an ellipse.
   The formal solution to this problem is called the Fisher linear discriminator. It maximizes the signal-to-noise ratio of y, where y is the projection of the input $y = \mathbf{w}.\mathbf{x}$.

5. Optional: Probabilistic interpretation of the logistic function. In general, Gaussian noise in more than one dimension is described by a covariance matrix with entries $C_{ij} = \langle (x_i - \langle x_i \rangle)(x_j - \langle x_j \rangle) \rangle$. We label the input stimulus with $s = \{a, b\}$. Now the probability distribution of is given by $P(\mathbf{x}|s=a) = \frac{1}{\sqrt{2\pi \det(C^{-1})}}\exp[-\frac{1}{2}(\mathbf{x} - \langle \mathbf{x}^a \rangle)^T C^{-1}(\mathbf{x} - \langle \mathbf{x}^a \rangle)]$, where C^{-1} is the inverse of C. Assume that both stimuli are equally likely, i.e. $P(s=a) = P(s=b)$.
   Show that $P(s=a|\mathbf{x})$ can be written as $P(s=a|\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}.\mathbf{x}-\theta)}$, express $\mathbf{w}$ and θ in $\langle \mathbf{x^a} \rangle$, $\left\langle \mathbf{x^b} \right\rangle$ and C. Use Bayes theorem which says that $P(s|x) = P(x|s)\frac{P(s)}{P(x)}$, with $P(x) = \sum_s P(x|s)P(s)$. This means that the logistic function, often used to model neurons, can be interpreted as a hypothesis tester.

4 Differentiation and extrema

Differentiation calculates the local slope of a function.The definition is

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Remember the following rules:

- $\frac{d}{dx}(f(x) + g(x)) = \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$ (sum rule)
- $\frac{d}{dx}[f(x)g(x)] = f(x)\frac{d}{dx}g(x) + g(x)\frac{d}{dx}f(x)$ (product rule)
- $\frac{d}{dx}f(g(x)) = \frac{df}{dg}\frac{dg}{dx}$ (partial differentiation)

These rules can be checked by filling in the definition. Higher order derivatives are defined by applying the same procedure in sequence, i.e. $\frac{d^2f}{dx^2} = f''(x) = (f'(x))'$. In practice you hardly ever need beyond second order derivatives.

Differentiation is linear in that $\frac{d}{dx}(f+g) = \frac{d}{dx}f + \frac{d}{dx}g$ and $\frac{d}{dx}(\alpha f(x)) = \alpha\frac{d}{dx}f(x)$.

4.1 Higher dimensions

Much the same rules apply when we have a function of multiple variables such as $f(x, y, z)$. As we proceed to higher dimensions we introduce the partial derivative which is written as.

$$\frac{\partial f}{\partial x}$$

It denotes differentiation w.r.t. x only. The difference with the regular differential will not be very important for us. However, for instance if $F(t) = f(x(t), y(t))$ then $\frac{\partial F}{\partial t} = 0$, but $\frac{dF}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$.

In higher dimensions we will in particular need the gradient. It is denoted with ∇ and is defined as

$$\nabla f(x_1, x_2, \ldots) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots\right)$$

The gradient is a vector with as many components as the function has variables we differentiate towards. An intuitive example comes from physics where $\mathbf{F}(x, y) = -\nabla V(x, y) = (-\frac{\partial V}{\partial x}, -\frac{\partial V}{\partial y})$, that is, the force in a (two)dimensional potential V, points in the negative direction of the gradient.

In this example the function V maps from two dimensions (x and y) to one dimension. More generally, there are also functions that map one dimension into three dimensions (a curve in 3D), from two to three (curved plane), and from three to three, etc.

4.2 Taylor expansion

An important application of differentiation is the Taylor expansion. It allows us to approximate a function in the neighbourhood of a known value using its derivatives

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0) + \ldots + \frac{1}{k!}(x - x_0)^k f^{(k)}(x_0) + \ldots$$

When $x_0 = 0$, you have $f(x) = f(0) + xf'(x_0) + \frac{1}{2}x^2 f''(x_0) + \ldots$

Some important ones (x_0 is assumed to be 0 and x is assumed small) are

- $\exp(x) \approx 1 + x + \frac{1}{2}x^2$
- $\log(1 + x) \approx x - \frac{1}{2}x^2$

- $\frac{1}{1+x} \approx 1 - x + x^2$

The use of these expansions is the following: When developing a model the expressions get rapidly much too complicated to analyze. One can then still try to see how the system will react to small changes in inputs by using these expansions. Of course, one should always check if the approximations made were valid. This can be done either in simulation or by calculating the size of the higher order terms.

As an example suppose we have a network that consists of two nodes in series, such that the output is $y = \tanh(\tanh(x))$. How does y vary with small changes of x around $x = 0$? Because $\tanh(x) = [\exp(x) - \exp(-x)]/[\exp(-x) + \exp(x)]$, $\tanh(x) \approx 2x/2 = x$. So around $x = 0$ we have $y \approx x$.

4.3 Extrema

Another important application of differentiation is the maximization or minimization of a function. At the maxima and minima of a (smooth) function the derivative is zero for all its dimensions. Just for ease of language we will assume we are looking for a minimum.

In the simplest cases one can find the minima explicitly by differentiating and solving when the derivative to be equal to zero. A minimum can be either global (truly the minimal value) or local (the minimum in a local region). For example, $f(x) = \sin x + x^2/10$ has many local minima but only one global minimum. If we have no analytical expression the local minima are relatively easy to find (numerically), but the global minimum is hard to find.

Relevance for cognition

> Also in the study of cognitive system, we often try to optimize functions. Just a few examples are: 'fitness' in evolution, energy consumption in the nervous system (how to develop an energy efficient brain), neural networks that minimize errors, and neural codes that maximize information. Often one tries to argue that due to the pressure of evolution the cognitive processes are the optimal solution to the problem at hand given the biological constraints. Another example we will explore below is the error function used to train neural networks.
>
> There are many such cost functions and it is not always obvious which one is the best choice (in case we want to build something) or most natural choice (in case we study the biology).

4.4 Constrained extrema: Lagrange multipliers

Sometimes we are looking for a minimum but under constraints. For instance we might want to maximize the output of a node, but the weights are constrained.

Suppose we are looking for the minimum of $f(\mathbf{x})$, under the constraint $g(\mathbf{x}) = a$. In some cases one can directly eliminate one of the variables. For instance we want the biggest area rectangle $f(\mathbf{x}) = x_1 x_2$ given that the sum of the height and the width is given, $x_1 + x_2 = L$. So we need to maximize $f(\mathbf{x}) = x_1(L - x_1)$, which is maximal if $x_1 = x_2 = L/2$.

If we cannot eliminate variables we can use Lagrange multipliers. The first step is to express the constraint as $c(\mathbf{x}) = 0$, so $c(\mathbf{x}) = g(\mathbf{x}) - a$ in this case. The trick is that we search for minima in $f(\mathbf{x}) + \lambda c(\mathbf{x})$, that is we solve

$$\frac{\partial}{\partial x_i}[f(\mathbf{x}) + \lambda c(\mathbf{x})] = 0$$

for all i. Given that $\mathbf{x}$ is n-dimensional there will be n such equations.

Example: We like to know what is the biggest rectangle that fits on a circle of radius r. If we assume that the circle is centered around $(0,0)$, the area of this rectangle is given by $f(\mathbf{x}) = 4x_1x_2$. The constraint fits on a circle and is written as $c(\mathbf{x}) = x_1^2 + x_2^2 - r^2$. So that we have to solve $\frac{\partial}{\partial x_1}[4x_1x_2 + \lambda(x_1^2 + x_2^2 - r^2)] = 4x_2 + 2\lambda x_1 = 0$ and $4x_1 + 2\lambda x_2 = 0$. The solution is $x_1 = x_2$ and $\lambda = -2$. The value of λ has no importance to us. But $x_1 = x_2$ tells us that a square is the best solution, as you might have expected.

4.5 Numerically looking for extrema

In high dimensional spaces, the search for extrema is using often done numerically. The problem can be compared to walking in a mountainous landscape with ridges, canyon and plains. As an example back-propagation networks (see below) are trying to minimize the difference between the actual output of the network and the desired output.

A simple, but not very good method is gradient descent. It simply means that at each point we follow the steepest way down. For instance we want to minimize $f(x,y) = x^2 + 2y^2$. The gradient is the vector $\nabla f(x,y) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = (2x, 4y)$. From a starting point (x_0, y_0) one determines the next point as $(x_1, y_1) = (x_0, y_0) - \eta\nabla f(x_0, y_0)$, and this is repeated until the vector no longer changes. The step size η can here be 0.1, but has to be chosen carefully in general. If you implement this algorithm you we see that it works quite well, however our function was particularly well-behaved.

This method can also be used when the function f is unknown and the gradient has to be calculated numerically.

Two problems occur:

1) The convergence should ideally be accurate but fast. Combining these two objectives is tricky: The search should not take too big steps, which could lead to overshooting the minimum. On the other hand too small steps will lead to slow performance. Many methods have been developed to deal with this problem [Press et al., 1988] (free online).

2) The other problem that one might encounter are local minima. The problem is like hill climbing in the mist: it is easy to reach a peak, but it is hard to be sure that you reached the highest peak. A simple trick is to restart the program with different starting conditions.

4.6 Exercises

1. Differentiate $\frac{d}{dx}\exp(-x^2)$, $\frac{d}{dx}\frac{1}{1+\exp(-x)}$, $\frac{d}{dx}\tanh(x)$, $\frac{d}{dz}zy^2$.

2. Calculate once using the sum rule and once partial differentiation $\frac{d}{dx}3x$. Calculate $\frac{d}{dx}x^2$ with the product rule and also with partial differentiation.

3. Differentiate $\frac{d}{dy}[A.\mathbf{v}(y)]$, where the matrix $A = \begin{pmatrix} 1 & 7 \\ 4 & 2 \end{pmatrix}$ and $\mathbf{v}(y) = \begin{pmatrix} y^2 \\ g(y) \end{pmatrix}$. Conclusion?

4. We try to minimize the cost of a cardboard box. The cost is $f(x,y,z) = 3xy+2xz+2zy$, the volume of the box is $V = xyz$. Using Lagrange multipliers derive the lowest cost solution for a given volume.

5. Sketch a few functions for which it is either very hard or very easy to find their minimum.

6. Money in the bank accumulates as $m(1+r)^n$, where m is the starting capital r is the interest and n is the number of years. The interest rate is low ($r \ll 1$). Derive an approximate expression for the amount after n years using a 2nd-order Taylor expansion (i.e. including terms r^2) around $r = 0$. Also estimate the error you make in this approximation. Check numerically for a few cases.

5 Application: The Back-propagation algorithm

We saw that the perceptron can not compute nor learn problems that are not linearly separable. The simplest example of this we have already encountered, namely the XOR problem. But a network with hidden layers can perform the computation. The network shown in Fig. 10 left, calculates the XOR function. Check that this network indeed implements the XOR function.

5.1 Multi-Layer perceptron

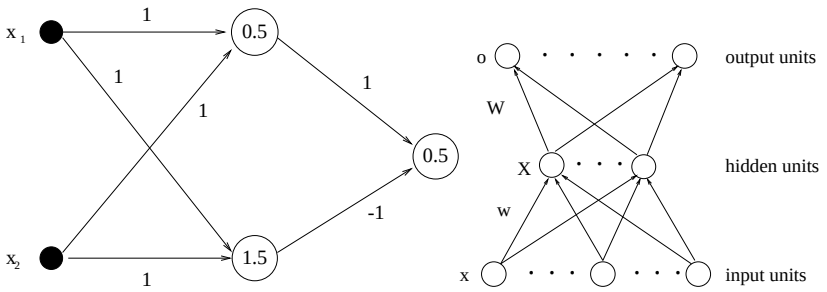

Figure 10: Left: A network that calculates the XOR function. Here the inputs are 0 or 1, and the open circles are binary threshold nodes with outputs 0/1. The numbers in the nodes are the thresholds for each unit. The rightmost node has output '0' when its total input is less than 0.5, the output is '1' otherwise.
Right: General layout of a multilayer perceptron network.

The important difference with the perceptron is that there is an extra layer of nodes, the so-called HIDDEN LAYER. The general layout of the multi-layer perceptron (MLP) network is in Fig. 10 (right). The nodes usually have a tanh or logistic transfer function.

We use the naming convention shown in Fig. 10. The weights between the layers are written as matrices. The output of node i is

$$\begin{aligned} o_i &= g(h_i) = g(\sum_j W_{ij} X_j) \\ &= g(\sum_j W_{ij}\, g(\sum_k w_{jk} x_k)) \end{aligned}$$

where we label the inputs with k and the hidden layers with j.

The task of the network is to learn a certain mapping between input vectors and output vectors. Note, both the input and the output can have many dimensions. For example, the task can be to see whether the digit is an '8', but we might for instance also want other outputs that indicate the size of the digit.

The computational power of these networks with one hidden layer is striking: A NETWORK WITH ONE HIDDEN LAYER CAN APPROXIMATE ANY CONTINUOUS FUNCTION! More than one hidden layer does not increase the class of problems that can be learned. Knowing this universal function approximation property of layered networks, the next question is how the train the network. We have now all the necessary ingredients to derive the back-propagation learning rule for neural networks. When the network gives the incorrect response we change the function of the network by adjusting its weights (and biases). We hope that after enough trials the learning rule converges (this is not guaranteed) and the network always gives the right answer. It is again a supervised learning algorithm: the desired outcome is known for every input pattern.

5.2 Back-propagation learning rule

To derive the learning rule we first need an error function or cost function, which tells us how well the network is performing. We want the error function to have a global minimum, to be smooth, and to have well-defined derivatives. A logical, but not unique choice is

$$error: \; E = \frac{1}{2}\sum_i (o_i - y_i)^2$$

where y_i is the desired output of the network and where o_i is the actual output of the network. Note that indeed this function has the required characteristics, it is smooth and differentiable. This is also the error that you minimize when you fit a straight line through some data points (see 'polyfit' in Matlab). The only way it can reach its minimal value of zero, is when $o_i = y_i$ for all i.

When tested for *all* training data the error becomes

$$E = \frac{1}{2}\sum_\mu \sum_i (o_i^\mu - y_i^\mu)^2$$

so if the error is zero for all patterns, we have $o_i^\mu = y_i^\mu$ for all patterns μ, in other words, the network works perfectly.

The learning adjusts the weights so as to reduce the value of E by checking if a small change in a particular weight would reduce E. It does this be calculating the derivatives of E w.r.t. the weights; it is therefore a so called gradient method. The trick will be not only to apply this to the weights connecting the hidden to the last layer (denoted with W), but also to the weight connecting the input layer to the hidden layer (denoted w). This gives the so-called ERROR BACK-PROPAGATION RULE.

Remember that $o_i = g(h_i) = g(\sum_j W_{ij} X_j)$, where $X_j = g(\sum_k w_{jk} x_k)$ and h_i is the net input to node i. The learning rule for the weights that connect the hidden layer to the output layer is $W_{ij} \rightarrow W_{ij} + \Delta W_{ij}$ with

$$\begin{aligned} \Delta W_{ij} &= -\eta \frac{dE}{dW_{ij}} \\ &= -\eta \frac{d}{dW_{ij}} \left(\frac{1}{2}[y_i - g(\sum_{j'} W_{ij'} X_{j'})]^2 \right) \\ &= \eta (y_i - o_i) g'(h_i) X_j \end{aligned} \tag{3}$$

where h_i is net input to output node i, that is $h_i = \sum_j W_{ij} X_j$. Again η is a small number which determines the learning rate, its value has to be determined with a bit of trial and error.[4]

We can rewrite Eq. 3 as

$$\Delta W_{ij} = \eta \delta_i X_j$$

with $\delta_i = g'(h_i)(y_i - o_i)$.

We thus have an update rule for the weights from hidden to output layer. For the weight connecting the input layer to the hidden layer we can play the same

[4]Note the difference between j which a given number and j' which is a sum variable. To do the differentiation it is best to write the sum out. For instance, we like to calculate $d(\sum_{j'} b_{j'} a_{j'})/db_2$, we have $d\sum_{j'} b_{j'} a_{j'}/db_2 = d(a_1 b_1 + a_2 b_2 + \ldots)/db_2 = a_2$. Or more formally, $d\mathbf{a}.\mathbf{b}/db_j = d(\sum_{j'} b_{j'} a_{j'})/db_j = \sum_{j'} \delta_{j,j'} a_{j'} = a_j$. In our case $\frac{d}{dW_{ij}}(\sum_{j'} W_{ij'} X_{j'}) = X_j$.

trick, although the maths is a bit more tedious $o_i = g(\sum_j W_{ij} g[\sum_k w_{jk} x_k])$:

$$\begin{aligned}
\Delta w_{jk} &= -\eta \frac{dE}{dw_{jk}} = -\eta \frac{\partial E}{\partial X_j} \frac{\partial X_j}{\partial w_{jk}} \\
&= \eta \sum_i (y_i - o_i) g'(h_i) W_{ij} \quad g'(h_j) x_k \\
&= \eta \sum_i \delta_i W_{ij} g'(h_j) x_k \\
&= \eta \delta_j x_k
\end{aligned}$$

with the *back-propagated* error term defined as $\delta_j = g'(h_j) \sum_i W_{ij} \delta_i = g'(h_j)(W^T \vec{\delta})_j$. This completes the back-propagation rule.

In a computer program applying the back-propagation would involve the following steps:

- Give inputs x_k to the network
- calculate the output and the error
- back-propagate the error: i.e. calculate δ_i and δ_j.
- calculate ΔW, Δw and the new weights

We have to repeat this procedure for all our patterns, and often many times until the error E is small. In practice we might stop when E no longer changes, because it can happen that the learning will not converge to the correct solution.

5.3 Comments

The back-propagation algorithm and its many variants are widely used to solve all types of problems: hand-writing recognition, credit rating, language learning are but a few. In a way the back-propagation is nothing but a fitting algorithm. It slowly adjusts the network until the input-output mapping matches the desired function. It has the nice property that it generalizes, even before unseen data will get a decent response. But like most fitting algorithms we encounter the following problems.

Convergence speed (how many trials to learn)

We want the error minimization to run quickly (few iterations), yet accurate. To prevent overshooting take a low enough learning rate. In simplest terms this is a trade-off between a small and large learning rate, but more advanced techniques exist, for instance one can add a 'momentum' term. This means that the algorithm remembers the direction of descent. This prevents 'swagging' around in the valley of optimality.

Local minima

The learning can get stuck in local minima, in that case the error remains high. I.e. convergence is not guaranteed! Whether this happens will depend on the initial conditions we have chosen for the weights. So if training fails, one should reset to other random initial weights and start again. Another option is to add some noise.

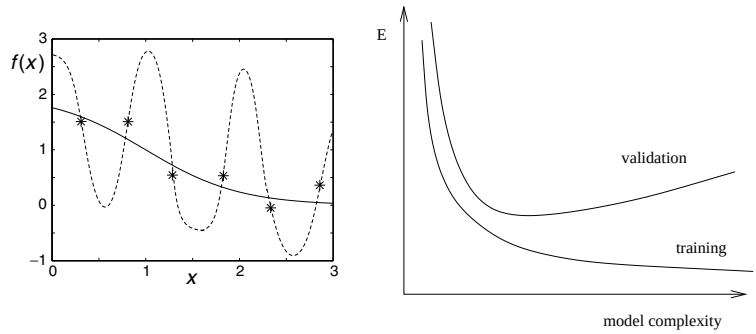

Figure 11: Over-fitting: Left: The stars are the data points. Although the dashed line might fit the data better, it is over-fitted. It is likely to perform worse on new data. Instead the solid line is a more reasonable model. Right: When you over-fit, the error on the training data decreases, but the error on new data increases. Ideally *both* errors are minimal.

Over-fitting

In most applications one is not simply trying to make a XOR network, but rather one is just given a dataset and the network's task is to give correct responses to new data by extracting rules and regularities from the data. Suppose the network has learned perfectly and after training zero error on any of data. This is good, however, how well does the network work for new data? To know this, one often trains the network with only half of the available data and tests the network with the remainder of the data after training. This testing with novel data is called validation of the model. When we use more and more weights and units and train for longer time, the network will describe the training data better and better. But the error on unseen data will likely increase! The generalization performance deteriorates, Fig. 11.

One trick is to add a weight decay term, this punishes extreme weights

$$E = \frac{1}{2}\sum_i (y_i - o_i)^2 + \alpha \sum_{i,j} w_{ij}^2$$

With the α term E becomes a mix of two terms. Learning causes the network to go towards a solution in which both the error and the weights are small. Note, that it is very unlikely that E will reach zero in this case.

This cost function prevents that output is generated by sensitive cancellation of large positive and negative weights, as a result the fitted function becomes smoother. More modern Bayesian techniques can also prevent such problems using a more fundamental approach.

Curse of dimensionality

As the number of inputs increases, the network becomes harder to train. It needs exponentially more data. The space grows very quickly. In high-dimensional space the samples are very sparse and lie far apart. This is a fundamental problem which is very hard to solve.

5.4 Properties of the hidden layer

The hidden layer can reveal hidden structure in the data. Suppose we create a network with limited number of hidden nodes. E.g. 100 input, 10 hidden, 100

output nodes. Now train the network such that output reproduces the input, for instance a small image, as good as possible. Because the hidden layer is smaller than the output and input, it will not do a perfect job. But to minimize the error, the units need to create independent representations, each representing some feature of the data. In other words the hidden layer finds an efficient representation of the data. Also technologically this has applications, after all the hidden layer compresses the information from 100 nodes into 10 nodes. One uses this to do image compression.

5.5 Biology?

Finally, it is good to note that it is not at all clear that the brain implements back-propagation, there is no evidence in favour of it nor against it. The presence of an error signal and the implementation of the back-propagation phase are problematic. Despite its questionable biological relevance the back-propagation is powerful algorithm, which has many applications. In recent years a couple of smarter algorithms have been developed. For engineers who are more interested in solving a task rather than model biology, those are usually preferable (see LFD and PMR courses).

5.6 Matlab notes

Matlab has a special toolbox to simulate neural networks. Try help and `nndtoc`, `nntool` on the command line. Programming the basic back-propagation algorithm yourself is not that difficult, however. There is a script on the website.

6 Filters

A coarse approximation of many sensory processes is to describe them as a linear filter. In order to analyze filters, complex numbers and functions are a useful tool.

6.1 Intermezzo: Complex numbers

Complex numbers arise when we try to solve quadratic equations. They arise also when dealing with periodic functions and Fourier transformation. Although complex numbers have quite interesting mathematical properties, we just use them as a tool here.

Consider the simple equation $x^2 = -1$. There is no real value for x which solves this, but we can define the solution. The solution is $x = i = \sqrt{-1}$, the solution is called imaginary. A complex number has in general a real and an imaginary component: $z = a + bi$. The real part is denoted $Re(z)$ or $\Re(z)$, and the imaginary part $Im(z)$ or $\Im(z)$, so that $Im(2 - 3i) = -3$. A lot of the rest follows from the simple rule that $i.i = -1$.

Addition of two complex numbers goes components-wise: If $z = x + iy$ and $c = a + bi$, $z + c = x + iy + a + bi = (x + a) + i(y + b)$. The multiplication of two complex numbers uses that $i.i = -1$. If $z = x + iy$ and $c = a + bi$, $zc = ax + aiy + bxi + byi^2 = (ax - by) + i(ay + bx)$. For instance, $z = 1 + 2i$, $c = 4 - 3i$, we have $z + c = 5 - i$, $z.c = 10 + 5i$.

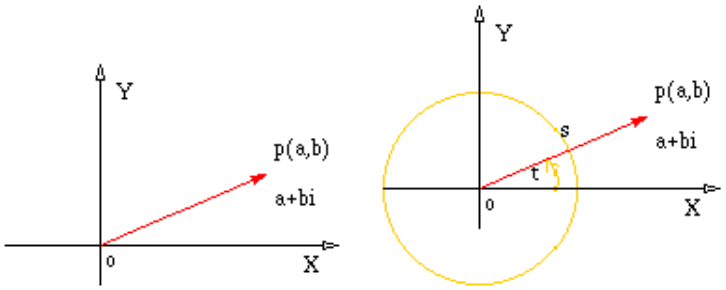

Figure 12: Complex plane. Left: representing the complex number p. Right: Using polar representation the complex number is defined by its argument (angle) and modulus (length).

Complex numbers can be drawn in the complex plane. The real part of a complex number is the x-coordinate, and the imaginary part the y-coordinate. The real and imaginary component of a number act a bit like the two components to a vector. To add two complex numbers, you just add them like you would add vectors.

Multiplication of two complex numbers also has a geometric representation in the complex plane. We first represent the complex numbers with the so called polar representation. (You can do the same for ordinary two dimensional vectors). Instead of giving the real and imaginary component, we specify the length $|z| = \sqrt{Re(z)^2 + Im(z)^2}$, called *modulus*, and the angle with the x-axis $\arg(z) = atan(Im(z)/Re(z))$, called *argument*.

Multiplication is simple in polar representation: The length of the product is the product of the lengths, and the argument of the product is the sum of the arguments. So if $d = zc$, then $|d| = |z||c|$ and $\arg(d) = \arg(z) + \arg(c)$. Use this to check that $(2i)^2 = -4$.

The COMPLEX CONJUGATE flips the sign of the imaginary component, it is denoted like $\bar{z}$. If $z = 1 + 2i$, then $\bar{z} = 1 - 2i$. The modulus can be written as

$|z| = \sqrt{\bar{z}z}$. (Check for yourself).

The division of two complex numbers should be done as follows: Suppose we want to calculate z/c, we know that the outcome will be again a complex number, say, $d = z/c$. Now multiply both numerator and denominator with $\bar{c}$, and you get $d = z\bar{c}/|c|^2$.

For example: $\frac{10+5i}{1+2i} = \frac{(1-2i)(10+5i)}{(1-2i)(1+2i)} = \frac{10+10-20i+5i}{1.1+2.2} = 4 - 3i$.

6.2 Complex functions

Most functions that you know (sin, sqrt, log) are defined for complex arguments as well. Most rules of integration, differentiation and simplification remain valid when the arguments are complex, for instance $\frac{d}{dt}\exp(i\omega t) = i\omega\exp(i\omega t)$. Most useful is the exponential function, $\exp(x+iy) = \exp(x)[\cos(y) + i\sin(y)]$, while the real part of the argument describes the exponential function, the imaginary component of the argument describes a periodic function.

In particular when dealing with periodic functions, the complex exponential function is particularly helpful in simplifying the calculations. But in the end after the calculation is done we throw away the imaginary part and we only use the real part of the function.

6.3 Temporal filters

In cognitive processing but also in data processing one often encounters FILTERS. Filters can be defined with a so called *kernel*, here labeled k. If the original signal is $f(t)$, the filtered signal $f^*(t)$ will be

$$f^*(t) = f * k = \int_{-\infty}^{\infty} f(t')k(t-t')dt'$$

This operation is called the CONVOLUTION of f with k, and is denoted with $*$; there is no standard notation for the filtered version of a function, here we use f^*. Note that, the filtering operation is linear (see also Chap.2), because $(\alpha f)^* = \alpha f^*$ and $(f+g)^* = f^* + g^*$. Furthermore, note that $f * k = k * f$, so that it is in theory unimportant what we call the kernel.

If only the output of the filter is known, the kernel can be retrieved by taking the input f a sharp, single peaked function. This is called a delta function $\delta(x)$. It is normalized such that $\int \delta(x) = 1$. Its most important property is that $\int_{-\infty}^{\infty} \delta(x - x_0)g(x)dx = g(x_0)$. So when we take $f(t) = \delta(t)$ the filtered version is $f^*(t) = k(t)$, in other words the filtered output is the kernel itself. The kernel is therefor also called the impulse response. As an example, a hand-clap can be used to determine all the echos in a room and can predict in principle the response to any kind of sound in that room (assuming sound behaves linearly).

For temporal filters it is not unreasonable to assume that the filter only has access to the past values of f. Those filters are called causal, and it means that $k(t < 0) = 0$. A simple kernel is $k(t) = \frac{1}{\tau}\exp(-t/\tau)$ if $t > 0$ and zero otherwise, Fig. 13. Here τ is the timeconstant of the filter. The longer τ is, the more strong filtering occurs.

6.4 Filtering periodic signals

The kernel of Fig. 13 implements a low pass filter. To show this we take the original signal a periodic function, and study the output. If the filter is a low pass filter, it should attenuate the low frequency signals less than the high frequency ones. In Fig. 13 the response to a variety of inputs is shown.

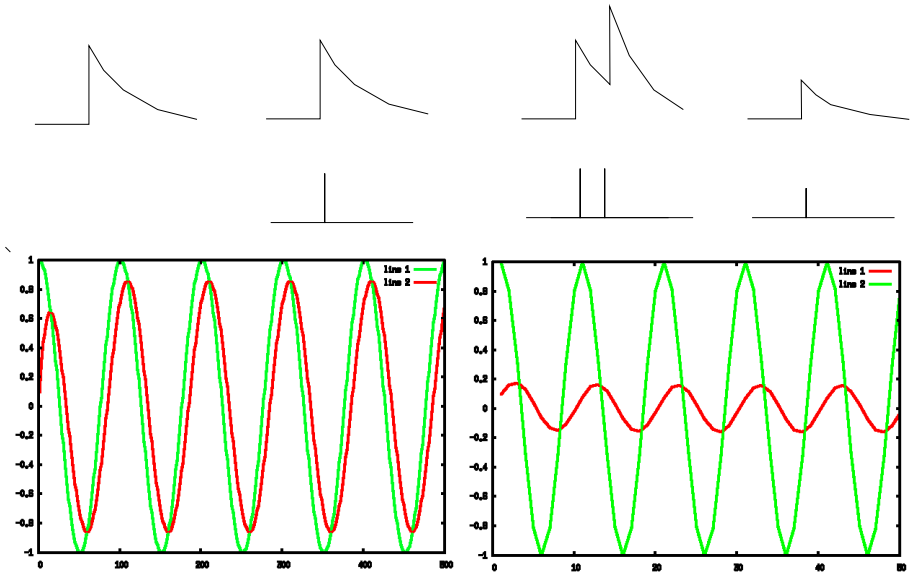

Figure 13: Left: the filter-kernel used. Middle left: the filter's response when the input is an impulse $\delta(t)$. Middle right: the response to two subsequent pulses. Right: Response to half the signal.
Below: The effect of filter on a periodic signal with a low frequency, the original signal is the largest, the filtered signal is slightly weakened and phase-shifted. Below right: The effect of filter on a periodic signal with a high frequency, the filtered signal is strongly weakened and more phase-shifted. (Note, the time scales left and right are different).

The easiest way to analyze how the filter above acts on periodic (sinusoidal) signals is to use complex numbers. The input signal is a periodic signal $f(t) = A\exp(2\pi ift)$, where f is the frequency of the signal, and A is its amplitude. The filter output is

$$\begin{aligned} f^*(t) &= \int_{-\infty}^{\infty} A\exp(2\pi ift')k(t-t')dt' \\ &= \frac{A}{\tau}\exp(-t/\tau)\int_{-\infty}^{t}\exp(2\pi ift'+t'/\tau)dt' \\ &= \frac{A}{1+2\pi if\tau}\exp(2\pi ift) \\ &= \frac{1}{1+2\pi if\tau}f(t) \end{aligned} \tag{4}$$

The outcome says that the output is equal to the input times a complex number. The ratio between the output amplitude and the input amplitude can be calculated as follows: $|f(t)|^2 = A^2\exp(2\pi ift)\overline{\exp(2\pi ift)} = A^2$, while,

$$\begin{aligned} |f^*(t)|^2 &= \left(\frac{1-2\pi if\tau}{1+(2\pi f\tau)^2}\right)\overline{\left(\frac{1-2\pi if\tau}{1+(2\pi f\tau)^2}\right)}|f(t)|^2 \\ &= \frac{(1-2\pi if\tau)(1+2\pi if\tau)}{[1+(2\pi f\tau)^2]^2}|f(t)|^2 \end{aligned}$$

So $\frac{|f^*(t)|}{|f(t)|} = \frac{1}{\sqrt{1+(2\pi f\tau)^2}}$. As the frequency increases the output amplitude diminishes, as a low-pass filter is supposed to do. Apart from the changing amplitude

ratio, the output signal changes it phase w.r.t. the input signal, Fig. 13.

In the end we only care about the real part of the solution. In principle we could have done the same calculation only using real functions, $f = A\sin(2\pi ft)$, but the work would have been more involved.

6.5 Spatial filters

In image processing and both human and computer vision, we can define spatial filters. Quite analogous to a temporal convolution we have a two dimensional convolution. The kernel of these spatial filters can be written as matrices. The input image is given as a matrix I with pixel intesities, the output image will be

$$I^*_{i,j} = \sum_k \sum_l I_{i-k,j-l} K_{k,l}$$

A kernel such as $K = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ will blur the image. The kernels do not need to be square. When dealing with a continuous, non-pixelated image the sums are trivially replaced by integrals.

(In Matlab and in GIMP you can define your own convolution matrix and filter images).

Relevance for cognition

Models of the processing in the visual cortex often include filters both in the spatial and the temporal domain. For instance, flicker frequencies higher than 50Hz are usually not visible by humans, meaning that the temporal cut-off is around 50Hz.

Spatially, many neurons in the primary visual cortex act as edge detectors and can be well described with a linear filter (in higher visual areas this breaks down). A (vertical) edge detector has a kernel such as $K = \begin{pmatrix} 1 & -1 \\ 2 & -2 \\ 1 & -1 \end{pmatrix}$.

There is evidence that different spatial frequencies are processed in parallel pathways.

In the auditory domain temporal filters are prevalent as well, as each neuron in the auditory cortex processes a limited frequency band.

6.6 Matlab notes

Given the enormous importance of filters in engineering, Matlab is well equipped to filter signals. The following list of commands was used to produce Fig. 13.

```
k=exp(-[0:0.1:5]); % exponential kernel
k=k/sum(k); % normalize such that sum(k)=1
f=0.1;s= cos(6.28*f*[0:0.1:100]) % define a low frequency signal 's'
c=conv(k,s); % the convolution command
plot(c(1:500))

f=1;s= cos(6.28*f*[0:0.1:100]); % repeat for a high frequency signal
c=conv(k,s);
plot(c)
```

In advanced applications convolutions are almost always done in Fourier space, as there the convolution becomes a multiplication. This procedure, including time to do the Fourier transform and the inverse transform, is much quicker.

6.7 Exercises

1. Check that $|\exp(x+iy)| = \exp(x)$.
2. Extract the real part from $f(t) = \exp(2\pi i f t)$, and from $g(t) = \frac{1}{1+2\pi i f \tau}\exp(2\pi i f t)$.
3. Given the function $f = 3$ if $0 < t < 4$ and zero otherwise. Take the kernel $k(t) = 1/2$ if $-1 < t < 1$ and zero otherwise, and filter f with it (either by hand or using Matlab). Sketch the result.
   To get a first intuition, calculate $f^*(t)$ when t is either much less or much larger than 0. Next, try $t = 2$.
4. As above, but now with $k(t) = -1/2$ if $-1 < t < 0$, $k(t) = 1/2$ if $0 < t < 1$.
5. Suppose we have a spatial filter with kernel $K = \begin{pmatrix} -1 & -2 & -1 \\ 1 & 2 & 1 \end{pmatrix}$. The input is zero everywhere except for a little patch where $I = \begin{pmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \end{pmatrix}$. Which normalized I ($\sum I_{ij}^2 = 1$) will give the maximal output of the filter? Note, it helps to think of both kernel and input as one-dimensional vectors.
6. How would you create a detector for longer lines, for lines with different angles? If you have GIMP, try these kernels on a sample image.

7 Differential equations

In previous chapters we concentrated on engineering and training networks to do a certain task. Here we approach cognitive computation differently, namely we ask how a system will behave given the basic building blocks. For instance, we might wonder how a network of biological neurons will react to a certain input. Many physical and biological systems are described in terms of differential equations. A differential equation is an equation which gives an expression for the derivative of a function. Here we mainly consider differential equations in time (again denoted t), although in general the derivatives can be w.r.t. any variable. The theory behind differential equations is quite involved, we just focus on a few cases without being too bothered whether solutions exist and if they are unique. Examples we will discuss here are an RC circuit, chemical reactions, rate neurons, and oscillations.

One of the simplest differential equations is the following one. Suppose we take a sausage roll with temperature T_0 into a room where the temperature is T_r. We are interested in the temperature over time $T(t)$. The energy flux will be proportional to the temperature difference between the room and the sausage roll. The temperate will therefor obey

$$\frac{dT(t)}{dt} = -c(T(t) - T_r)$$

where c is some cooling constant determined by insulation, the units should 'per second' (why?). This equation has the solution $T(t) = T_r + (T_0 - T_r)\exp(-ct)$. It is good that this solution is correct by substitution in the differential equation. Furthermore, you should check that $T(0) = T_0$ and $T(\infty) = T_r$.

We just went through the basic steps in studying a physical system using differential equations: 1) Write down the differential equation(s) that describe or approximate the physical system, 2) Solve the equation, 3) Add the initial conditions to solve any constants in the solution.

7.1 RC circuit

A very similar differential equation results from a simple electronic circuit, a so called RC circuit. The RC circuit is not only of interest for engineers but it is also the basis for most neuron models.

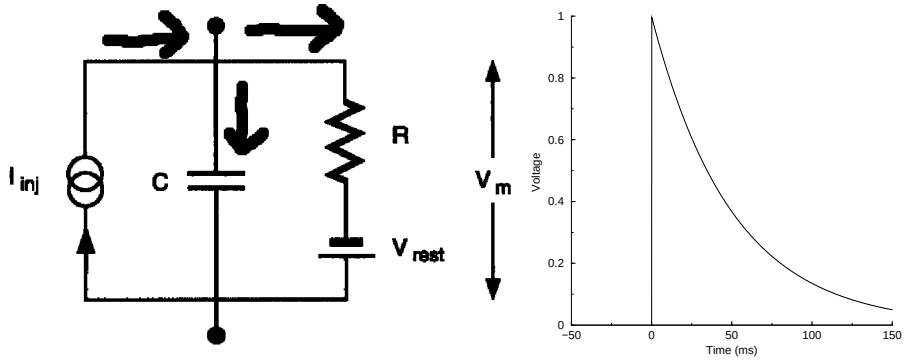

Figure 14: Left: RC circuit. The injected current is I_{inj}, the total capacitance is C and R is the total membrane resistance. Right: Response of the voltage to an impulse in the stimulus current.

Kirchhoff's law tells us that the sum of the currents at any point in the circuit

should be zero. What are the different contributions to the current? The current through the resistor is given by OHM'S LAW[5]

$$I_{resistor} = \frac{\Delta V}{R} = \frac{V - V_{rest}}{R}$$

Similarly, there is a current associated to the capacitance. This current flows for instance when initially the voltage across the capacitor is zero, but suddenly a voltage is applied across it. Like a battery, a current flows only until the capacitor is charged up (or discharged). The current into the capacitor is

$$I_{cap} = C\frac{dV}{dt}$$

It is important to note that no current flows when the voltage across the capacitor does not change over time.

Understanding electrical current: In order to get intuition for these equations, you can make the analogy with water. The height of the water corresponds to the voltage. The current to the amount of water flowing per second. A resistor in this language is a narrow tube that hinders the water flow. A capacity is a reservoir. The amount of current needed to make the water rise in a reservoir, will depend on its capacity.

Finally, we assume an external current is injected. As stated the sum of the currents should be zero. We have to fix the signs of the currents: we define currents flowing away from the point to be negative. Now we have $-I_{resistor} - I_{cap} + I_{ext} = 0$. The circuit diagram thus leads to the following differential equation for the membrane voltage.

$$C\frac{dV(t)}{dt} = -\frac{1}{R}[V(t) - V_{rest}] + I_{inj}(t)$$

In other words, the membrane voltage is given by a first order differential equation. It is always a good idea to study the steady state solutions of differential equations first. This means that we assume I_{inj} to be constant and $dV/dt = 0$. We find for the membrane voltage $V_\infty = V_{rest} + RI_{inj}$.

How rapidly is this steady state approached? If the voltage at t=0 is V_0, one finds by substitution that $V(t) = V_\infty + [V_0 - V_\infty]\exp(-t/\tau)$. So, the voltage settles exponentially. The product $\tau = RC$ is the time constant of the circuit. The time-constant determines how fast the membrane voltage reacts to fluctuations in the input current.

The equation describes actually the low-pass filter we have seen above, the voltage is a low-pass filtered version of the input current. Again this is easy to show using complex functions. Without going into the details we state that when $I_{inj} = I_0\exp(2\pi i f t)$, the solution is $V(t) = \frac{1}{1+2\pi i f\tau}RI(t)$ plus terms from the initial conditions. This can be compared to Eq. 4. The rate will follow the input with some delay. Like above, the amplitude of the voltage gets smaller at higher frequencies as $1/\sqrt{1+(2\pi f\tau)^2}$, i.e. the signal is low-pass filtered.

7.2 Rate model of a neuron

In neural and cognitive modelling one commonly models the neuron with its firing rate. Earlier we described the firing rate of a neuron $r(t)$ given some input as

[5]Ohm's law says that current and voltage are linearly related. As soon as the linearity is lost, Ohm's law is broken. This happens for instance in diodes or neon tubes, in that case we have a non-Ohmic conductance.

$r(t) = g(input(t))$. However, in biology the neuron will take time to establish its response. A common way to describe this is

$$\tau \frac{dr(t)}{dt} = -r(t) + g(\text{input}(t)) \tag{5}$$

where $g(input)$ is a rectifying function, such as $\max(input, 0)$ or a sigmoid such as $1/(1 + \exp(-x))$. The τ describes the time constant of the neuron (a few ms).

If the input is constant, the solution to Eq. 5 is $r(t) = c\exp(-t/\tau) + g(input)$. One can check this by substituting the solution in the differential equation. In words, the activity of the node adjusts to the new input. However, like the RC circuit the change is not instantaneous but has some sluggishness.

This more biological neuron model will not improve the performance of the perceptron or the back-propagation network, but it can be used to create networks that oscillate and for instance describe motor circuit responsible for the movement of limbs.[6]

7.3 Harmonic oscillator

Another commonly encountered example of a differential equation is a harmonic oscillator. Suppose we have a mass connected to a spring. The force of the spring is $F = -Kx$, where K is Hooke's spring constant and x is the position. The minus sign occurs moving x to the left will cause a force to the right. Because $F = ma = m\frac{d^2x}{dt^2}$, we end up with

$$m\frac{d^2x}{dt^2} = -Kx \tag{6}$$

Its solutions are periodic functions, $x(t) = A\cos(\omega t) + B\sin(\omega t)$, where $\omega = \sqrt{K/m}$. A very similar differential equation can be derived from a pendulum with small amplitude.

Eq. (6) can also be written as two 1st order equations by introducing the variable v for speed

$$\begin{aligned} v(t) &= \frac{dx(t)}{dt} \\ m\frac{dv(t)}{dt} &= -Kx \end{aligned}$$

This is a general trick, any higher order derivative can be replaced with first order derivatives, at the cost of introducing extra variables. The number of 1st order equations thus obtained is called the order of the differential equation.

7.4 Chemical reaction

Also chemical reactions can be described with differential equations. This is for instance useful if we want to build low-level models of neurons. Suppose we have two chemicals A and B. We denote their concentration with $[A]$ and $[B]$. The reaction from A to B occurs with a reaction rate k_{ab} and the reverse reaction with a rate k_{ba}. These reactions can be written as

$$\begin{aligned} \frac{d[A]}{dt} &= -k_{ab}[A] + k_{ba}[B] \\ \frac{d[B]}{dt} &= -k_{ba}[B] + k_{ab}[A] \end{aligned} \tag{7}$$

[6] The spiking behavior of neurons, as shown in Fig. 5 (middle), can not be described by an RC circuit. It is described by the so-called Hodgkin-Huxley equations. These equations are non-linear and have 4 dimensions. The has no analytical solution, but it can be solved numerically. See [van Rossum,] for more details.

The first equation describes that the rate of change in A is caused by two processes: $-k_{ab}[A]$ describes A being lost because it is converted into B, whereas $k_{ba}[B]$ describes new A being produced from B.

We have here a simple two-dimensional differential equation. We can easily solve it by introducing two new variables $\Sigma(t) = [A](t) + [B](t)$ and $\Delta(t) = k_{ab}[A](t) - k_{ba}[B](t)$. Now it is easy to see that $d\Sigma(t)/dt = d[A]/dt + d[B]/dt = 0$, that is, Σ is constant. This reflects that no matter is lost in the reactions. The remaining equation for Δ is simple (try). By this substitution we have reduced the two-dimensional equation to a one-dimensional one. We could have done this already at the beginning: because $[A] + [B] = const$, we can eliminate $[B]$ in favor of $[A]$ and the constant, creating a one-dimensional equation. (try this as well)

7.5 Numerical solution

Solving differential equations analytically is often impossible. But solving differential equations numerically can be tricky as well, in particular when they are nonlinear and involve many variables. In practice one can resort to numerical standard routines. Here we show how to use the Euler method. The Euler method is the simplest way of integrating differential equations. Suppose we have a diifrerential equation

$$\frac{df(t)}{dt} = g(f(t), t)$$

where $f(t)$ and $g(f(t), t)$ are arbitrary functions. For instance, g can describe the combination of the drive and the decay of the system, see Eq. 5. It is important to realize that a differential equation has often a whole set of solutions. It is necessary to know the starting values; the initial conditions determine the value of the functions at $t = 0$.

When we integrate the equation we want to know the value in future time, given the value now. According to the definition of the derivative

$$\frac{1}{\delta t}[f(t + \delta t) - f(t)] = g(f(t), t)$$

or $f(t + \delta t) = f(t) + \delta t\, g(f(t), t)$. This is directly implementable in Matlab and would look something like this

```
f= 2 % a given initial value of f
for time=0:dt:endtime
f+= dt*g
end
```

The δt should be a small number such that f only changes little per time-step.

In practice the Euler method is so easy that I find it worthwhile, but is not very efficient and can be unstable. One can decrease or increase the step size to see if the solution remains the same. Alternatively, you can use Matlab's own routines, which often adapt the time-step so that you get a quick and accurate solution. There is a whole set of routines called ode, such as ode45.

7.6 Exercises

1. Write Eqs. (7) as a matrix equation. Hereto introduce a vector $\mathbf{s} = ([A], [B])$, and write $\frac{d}{dt}\mathbf{s} = M.\mathbf{s}$. Calculate the eigenvalues and eigenvectors of M. Suppose a certain $\mathbf{s}_1$ is an eigenvector of the M matrix, what is the differential equation for $\mathbf{s}_1$? Interpret your results.

2. Consider the differential equation: $b^2 \frac{d^2 f(x)}{dx^2} = -f(x)$. Plug in $f(x) = A\exp(Bx)$. For which values of A and B is this a solution? Write the solution as a periodic function. Check your solution by filling this into the differential equation.

8 Stability and Phase plane analysis

In this chapter we study how networks of neurons behave that are described with differential equations. In doing so we come across a potential implementation of working memory.

8.1 Single neuron

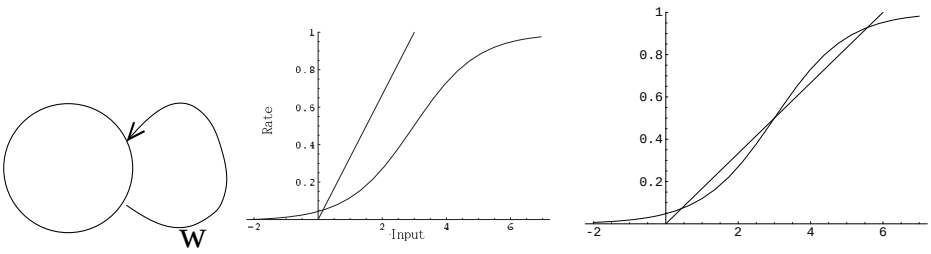

Figure 15: Left: A single neuron with a recurrent connection with strength w. Middle: rate as a function of the input, and the line $r = in/w$. The only solution is where the curves intersect. Right: For stronger w there are multiple solutions, the middle one is unstable, the two outer ones are stable.

Let's start simple. Suppose we have just a single neuron and we do not care about its dynamics, i.e. we do not care about its differential equation. The neuron does not receive any external input, but it does receive recurrent input with strength w, Fig. 15 left. The rate of neuron for a given input is here modelled as $r(in) = 1/(1 + \exp(-in + 5/2)$. The recurrent input is given by $in(r) = wr$, so that we need to find values for r that satisfy $r = 1/(1 + \exp(-wr + 5/2)$. This is impossible to solve analytically. We can solve this equation graphically by plotting both $r(in) = 1/(1 + \exp(-in + 5/2)$ and $r(in) = in/w$ in one graph, Fig. 15.

Depending on the value w we can have one or two stable points. When w is small (or absent) the rate is just small as well. When w is large enough, we have two stable points, one with low activity and one with high activity, in which the feedback keeps the neuron going. We have basically a flip-flop memory! Which state the neuron will reach depends on the initial conditions. Once the neuron is in one state, it will stay there, until we disturb it with external input. This 'network' acts as a single bit of working memory. (Working memory is the typ of memory used to keep items briefly in store. In contrast to long term memory which is thought to be implemented in the synaptic weights, working memory is thought to be activity based).

When w is even larger, the only fixed point is near $r = 1$. You can see this by imagining an even shallower line in Fig. 15.

8.2 Damped spring

How do we analyse a complicated system of differential equations? Using a phase plane we can sometimes get a better insight. Consider again the mass with the spring connected to it. We add a damping term which describes the effect of friction. Its sign is negative to express that the force is opposite the direction of the velocity.

$$m\frac{d^2x(t)}{dt^2} = -Kx(t) - c\frac{dx(t)}{dt}$$

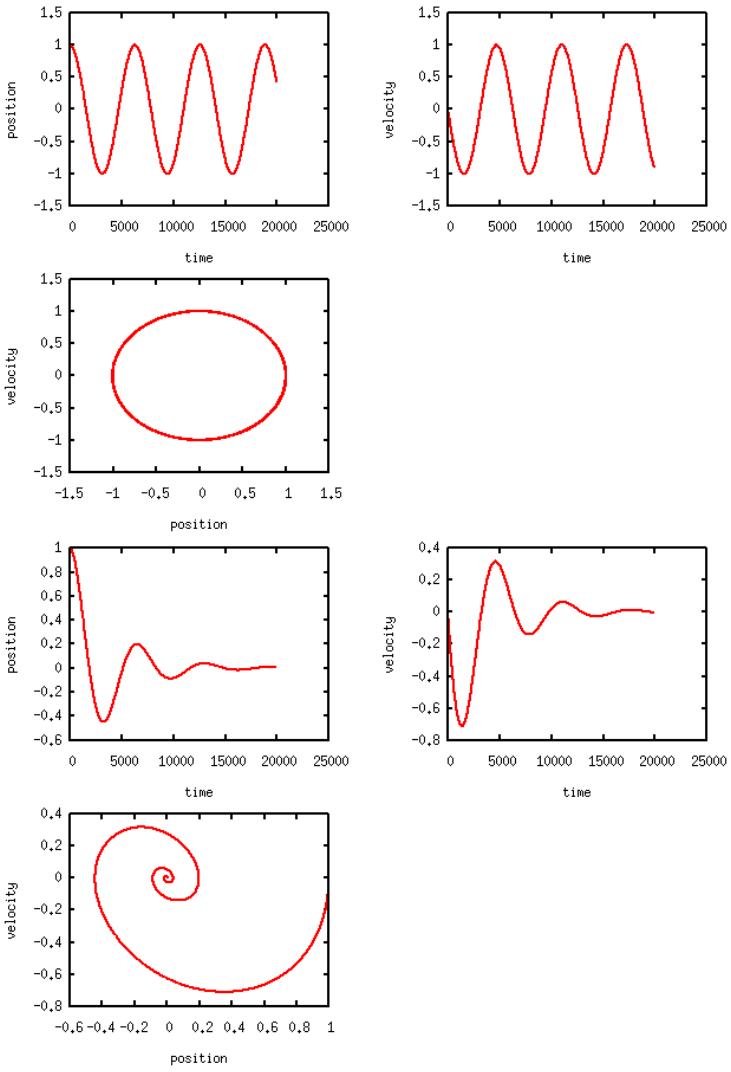

Figure 16: The oscillations in a spring and mass system.
Two top panels: The position and velocity versus time. Below that: the phase plane. Upper three plots: undamped case ($c = 0$).
Lower three plots: damped case ($c > 0$). The initial conditions were $x(0) = 1$, $v(0) = 0$.

We rewrite this as a system of two first order differential equations, by introducing the velocity as

$$\begin{aligned} \frac{dx(t)}{dt} &= v(t) \\ m\frac{dv(t)}{dt} &= -Kx(t) - cv(t) \end{aligned}$$

In PHASE PLANE we plot one state variable against another. Here we plot the position variable against the velocity, Fig. 16. Without damping the position and velocity are both sinusoidal, but out of phase. In phase space they run around in an ellipse.

When the damping is present, the oscillations die out. The final velocity and position are both zero in that case. We call these values of position and velocity

$(0,0)$ a STABLE FIXED POINT of the dynamics. No matter how strongly we perturb it, eventually this system will end up in the stable fixed point.

8.3 Stability analysis

The definition for a fixed point is that at the fixed point the temporal derivatives are zero. In the above the fixed point was stable. Not all fixed points are stable. Here we look at a method to determine whether a fixed point is stable or not. Consider two recurrently connected neurons: $u_1 \rightleftarrows u_2$

$$\begin{aligned} \tau \frac{du_1}{dt} &= -u_1 + [w_{12}u_2 + in_1]_+ \\ \tau \frac{du_2}{dt} &= -u_2 + [w_{21}u_1 + in_2]_+ \end{aligned} \tag{8}$$

where $[x]_+$ is a rectification with $[x]_+ = x$ if $x > 0$ and $[x]_+ = 0$ otherwise. This describes two neurons that provide input to each other. If $w_{12} > 0$ ($w_{12} < 0$) then neuron 2 has an excitatory (inhibitory) influence on neuron 1. Apart from that they receive external input (in_1, in_2) which is assumed to be constant.

Let's assume (to be confirmed post hoc) that $w_{21}u_2 + in_1$ and $w_{12}u_1 + in_2$ are much larger than 0, so that we can ignore the rectification. In that case we have

$$\begin{aligned} \tau \frac{d\mathbf{u}}{dt} &= \begin{pmatrix} -1 & w_{12} \\ w_{21} & -1 \end{pmatrix} \mathbf{u}(t) + \mathbf{in} \\ &= W.\mathbf{u}(t) + \mathbf{in} \end{aligned}$$

Let's find the fixed points, that is the $\mathbf{u}$ for which $\tau \frac{d\mathbf{u}}{dt} = 0$, we can solve this directly and write the fixed point as $\mathbf{u}_{fp}$, with $\mathbf{u}_{fp} = -W^{-1}.\mathbf{in}$. For convinience we set the input equal to $\mathbf{in} = (1,1)$. In Fig. 17 the resulting fixed points are right in the middle of the plot (indicated with a circle on the left and a cross on the right).

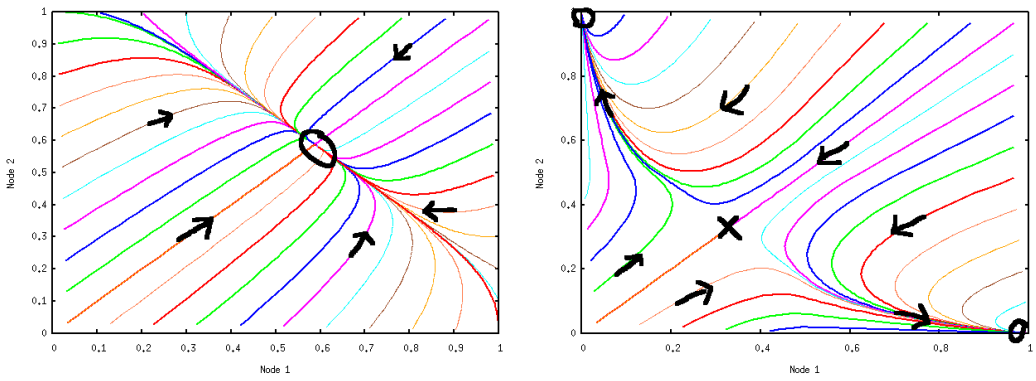

Figure 17: Stability of two connected nodes. On the left the weights are $w_{12} = w_{21} = -4/5$. There is a stable fixed point at $(0.59, 0.59)$. The eigenvalues of the stability matrix are -1/5 and -4/5.
Right: stronger mutual inhibition ($w_{12} = w_{21} = -2$). Now the fixed point at $(1/3, 1/3)$ is unstable. The eigenvalues are 1 and -3.

Next, we perform a stability analysis to see if the fixed point is stable. To this end we look at what happens if we perturb the system away from the fixed point, i.e. $\mathbf{u} = \mathbf{u}_{fp} + \delta\mathbf{u}$. Now $\tau \frac{d\mathbf{u}}{dt} = W.(\mathbf{u_{fp}} + \delta\mathbf{u}) + \mathbf{in} = W.\delta\mathbf{u}$, where $\delta\mathbf{u}$ is a small vector. The only thing we need to know if such a perturbation grows or shrinks

over time. An easy way is to perturb in the direction of the eigenvectors W. An eigenvector of W will behave as $\tau \frac{d\mathbf{s_i}}{dt} = \lambda_i \mathbf{s}_i$, and the perturbation will therefore develop as $s_i(t) = c.\exp(\lambda_i t/\tau)$. The sign of λ_i will determine whether the system runs away from the fixed point or returns to it: if $\lambda < 0$ than the perturbation will die away and the system will return to the fixed point, if $\lambda > 0$ the pertubation will grow. We can now distinguish a few possibilities.

- $\lambda_{1,2} < 0$. The dynamics are stable, the system converges to fixed point. This is illustrated in Fig. 17 left. The figure illustrates the system's evolution. We simulated Eq.8, and followed the system over time. Different sets of the initial conditions were taken, all along the edges of the graph.

- $\lambda_1 > 0$, $\lambda_2 < 0$. Saddle point. Although the dynamics are stable in one direction, in the other direction it is unstable. Therefore the fixed point as a whole is unstable. This is illustrated in Fig. 17 right. Along the diagonal the dynamics moves towards the fixed point $(\frac{1}{3}, \frac{1}{3})$, but then bends off towards to $(0, in_2)$ or $(in_1, 0)$. Once it hits the x-axis or the y-axis, the rectification that we ignored earlier kicks in. Without the rectification the activity would continue to grow to $(\pm\infty, \mp\infty)$. The intuition is that compared to the previous case, because the inhibition is stronger, the nodes strongly inhibit each other and there can only be one winner.

- $\lambda_{1,2} > 0$. Dynamics are unstable. This means that a minuscule fluctuation will drive the solution further and further from the equilibrium. Like in the previous case, the solution will either grow to infinity or till the linear approximation breaks down.

- If the eigenvalues are complex the system will oscillate. Remember: $e^{x+iy} = e^x[\cos(y) + i\sin(y)]$. Stability determined by the real part of the eigenvalue $Re(\lambda)$. When the real part is < 0 the oscillations die out, otherwise they get stronger over time. The damped spring system, Fig. 16 is an example of such a system.

The above technique also can be applied to the case when the equations are non-linear. In that case the fixed points usually have to be determined numerically, but around the fixed point one can make a Taylor expansion, so that for small perturbations $\tau \frac{d\mathbf{u}}{dt} \approx W.\delta\mathbf{u}$ and one can study the eigenvalues of W again.[7]

In Fig. 17 left, the system will always go to the same fixed point. The BASIN of attraction in this case encompasses all possible initial conditions. In Fig. 17 right we have two basins of attraction, starting above the line $in_1 = in_2$ the system will go to the upper left fixed point, starting below the line the system will go to the lower right fixed point,

8.4 Chaotic dynamics

If we connect a decent number of nodes to each other with random weights, we can get chaotic dynamics. If a system is chaotic it will show usually wildly fluctuating dynamics. The system is still deterministic, but it is nevertheless very hard to predict its course. The reason is that small perturbations in the initial conditions can lead to very different outcomes and trajectories (The butterfly in China, that causes rain in Scotland). This contrasts with the non-chaotic case, where small perturbations in the cause small deviations in the final outcome. On the website there is a script which allows you to play with a chaotic system.

[7] The other two fixed points in the corners of Fig. 17 right can be found using this technique. The Taylor expansion of the transfer function we used is unfortunately ill-defined near $x = 0$. Replacing it with $g(x) = \log(1 + \exp(10x))/10$, which is very similar, will work better.

There has been much speculation, but not much evidence for possible roles of chaotic dynamics in the brain.

8.5 Exercises

1. a) Check by graphical construction that in Fig. 15, there is only one solution when w becomes very large.
   b) How would the analysis of section 8.1 change if $r(in) = 1/(1 + \exp(-in)$?

2. Another way look at section 8.1 is to analyze the differential equation

$$\tau \frac{dr(t)}{dt} = -r(t) + g(wr(t))$$

   Without doing the actual calculation, 1) write down the equation for the fixed point, 2) indicate how stability of a given fixed point can be researched.

9 Application: Hopfield network

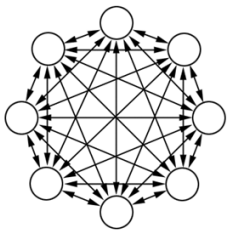

Figure 18: Hopfield net with 8 nodes. The arrows denote the (symmetric) weights between them.

We consider again a network of neurons. Importantly, we assume that the connections between the nodes are symmetric, that is, the weight from i to j is the same as the weight from j to i, i.e. $w_{ij} = w_{ji}$. In this case one can show that the dynamics is simple: the system always goes to one of its equilibrium states and stays there.

This type of network is called a Hopfield network and it is an influential memory model. The Hopfield network can store multiple binary patterns with a simple learning rule. This is called an AUTO-ASSOCIATIVE MEMORY: Presenting a partial stimulus leads to a recall of the full memory, see Fig. 19. Auto-associative memories are very different from computer memory (a bit like Google ...).

9.1 Single memory

Suppose we have n nodes in the network; each pattern is given by a n-dimensional vector $\mathbf{p}^\mu$ with binary entries (± 1), where μ labels the pattern. To see how the retrieval of the memory pattern works, suppose first just a single pattern is stored. For a single pattern the weights should be set $w_{ij} = p_i^\mu p_j^\mu$. The updating of the network should be done asynchronously. That is, each time-step pick a random node and update it according to $s_i(t+1) = g(w_{ij}s_j(t))$, where $g(x) = sign(x)$. We do this until the state of the network no longer changes.

Suppose just one node s_i is incorrect, all the other nodes have $s_i = p_i$. When we update the incorrect node we have $s_i(t+1) = sign(\sum_j w_{ij}p_j) = sign(\sum_j p_i p_j^2) = sign(Np_i) = p_i$, i.e. s_i is recalled perfectly. This was of course a simple task, but is not hard to see that the memory will be recalled even when up to half of the other bits is wrong.

If more than half of the bits are wrong, the system will go to the state $-\mathbf{p}$. By learning $\mathbf{p}$, the network automatically learn these flipped state as well.

9.2 Energy minimization

The network will always settle in a stable state. The network will evolve to a minimum of the following ENERGY-FUNCTION

$$E = -\frac{1}{2}\sum_{i,j}^{n} w_{ij}s_i s_j$$

To prove this, suppose we update s_i. It's new value is $s_i(t+1) = sign(\sum_j w_{ij}s_j)$. If $s_i(t+1) = s_i(t)$, the energy is the same, else $s_i(t+1) = -s_i(t)$ and

$$\Delta E = -\frac{1}{2}\sum_j w_{ij}s_i(t+1)s_j(t) + \frac{1}{2}\sum_j w_{ij}s_i(t)s_j(t) = -s_i(t+1)\sum_j w_{ij}s_j(t) < 0$$

The last inequality holds because $s_i(t+1) = sign(\sum_j w_{ij} s_j)$. Hence energy always decreases or stays, it never goes up. In other words, the network has a stable fixed point. This fixed point corresponds to the memory state.

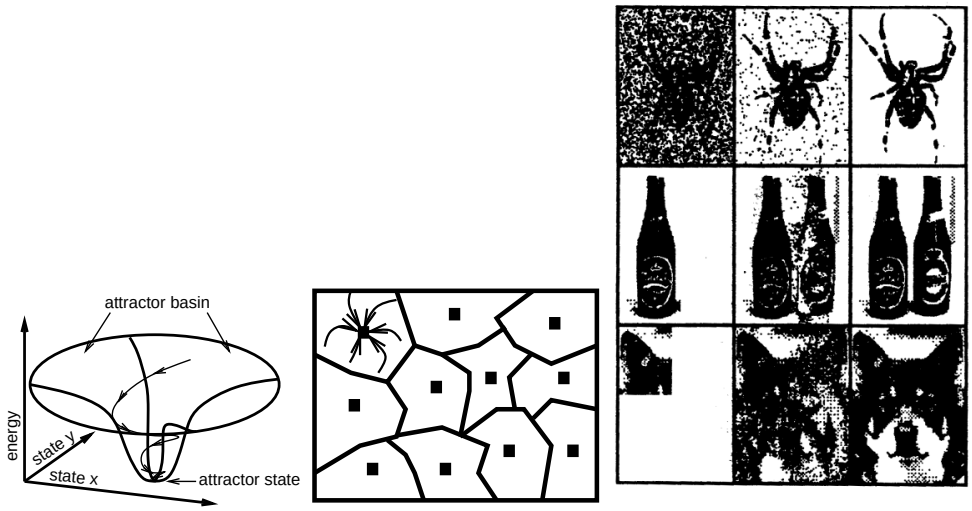

Figure 19:
Left: Whenever the network starts in a state close enough to an attractor, it will 'fall in the hole' and reach the attractor state.
Middle: Multiple attractor are present in the network, each with their own basin of attraction. Each corresponds to a different memory.
Right: Pattern completion in a Hopfield network. The network is trained on the rightmost (binary) images. Each time the leftmost (distorted) input is given, the network evolves via the intermediate state to the stored state. These different image can all be stored in the same network. From Hertz.

9.3 Storing many patterns

When multiple memories are to be stored, the weight between node i and j should be set according to the rule $w_{ij} = \sum_\mu p_i^\mu p_j^\mu$. This means we just sum the weights over all patterns. It also helps to force $w_{ii} = 0$ (hack). Each stored pattern will correspond to a stable fixed point, or attractor state. We can store multiple memories in the net as is shown in Fig. 19 right.

However, the storage capacity of the network is not unlimited. As we increase number of stored patterns, the memories states becomes unstable. Stable mixed states appear. At the critical amount of storage, performance decreases suddenly. The network will still equilibrate, but it will end up in spurious attractor states rather than in the memory states it was suppose to find. The numbers of patterns we can store is proportional to the number of nodes, $n_{stored} = \alpha n$ where α is called the capacity. Simulation and statistical physics gives $\alpha_c n = 0.138n$. Hence a network with 100 nodes can store about 14 patterns.

9.4 Biology and Hopfield nets?

There is no direct evidence for Hopfield like attractors in the brain. For instance, object recognition has not been observed as reaching an attractor. On the other hand, working memory states exists, which are likely due to an attractor network. The assumptions we made need further investigation

- Are connections in the brain symmetric? This is not well known, but definitely not all connections are symmetric on a cell to cell basis. However, some asymmetry does not destroy attractor states.
- Learning is one-shot and in order for the network to work, learning should stop after the to be learned patterns have been presented, otherwise network will start to overflow. One can get around this restriction by implementing weight decay in which old memories are slowly forgotten.

9.5 Exercises

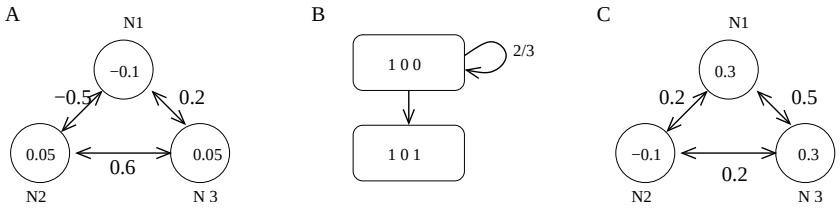

Figure 20: See exercise.

1. Think of examples of small networks where the dynamics does not settle down because 1) update is synchronous, or 2) weights are not symmetric.
2. Mini Hopfield network. Our first network is shown in Fig. 20A. Let us suppose that the neurons start off with the values N1 = 1, N2 = 1, N3 = 0. The triad '110' of values describes the network's state. A neuron becomes active (1) if the weighted sum of its inputs from the other neurons is larger than the threshold values, otherwise it will be inactive (0). [This is completely equivalent to the case where the inactive state is -1]. The weights are the values on the lines joining the neurons and thresholds are the values within the circles. Look at N1 first. The weighted sum of its inputs from N2 and N3 is 0.5 x 1 + 0.2 x 0 = -0.5. This is not bigger than the threshold of -0.1 so N1 switches off and the network moves into the state '010'. Starting from the original network conditions, what happens to N2 and N3?
   The Hopfield network neurons update asynchronously, i.e. not all at the same time. Each of the neurons has an equal chance of being updated in any given time interval. If it is updated, then a state transition can occurs. Draw a state transition diagram showing all the possible states of the network (8 in this case, 2 possible values for each of the three neurons) linked with arrows depicting the transitions that can occur to take one state into another. This is shown in Fig. 20 B, for the '100' state. In two-thirds of the cases, N1 is above threshold and the input to N2 is below threshold: in neither case do they change state. In the other case, when N3 tries to fire, its input is above threshold so it changes state from 0 to 1. Complete the state diagram for this network. What is special about the state 011?
3. Draw the state diagram for the network in Fig. 20C. What's the obvious difference between this state diagram and the previous one?

References

[Boas, 1966] Boas, M. L. (1966). *Mathematical Methods in the Physical Sciences.* Wiley, New York.

[Greenberg, 1998] Greenberg, M. D. (1998). *Advanced Engineering Mathematics*. Prentice Hall, New York.

[Hertz et al., 1991] Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Perseus, Reading, MA.

[Press et al., 1988] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1988). *Numerical Recipes in C*. Cambridge University Press, Cambridge. Fully online.

[Trappenberg, 2002] Trappenberg, T. P. (2002). *Fundamentals of computational neuroscience*. Oxford.

[van Rossum,] van Rossum, M. C. W. Lecture notes Neural Computation. Available on homepages.inf.ed.ac.uk/mvanross.