A brief introduction to probabilistic machine learning with neuroscientific relations

Thomas Trappenberg

Abstract My aim of this article is to summarize in a concise way what I consider the most important ideas of modern machine learning. I start with some general comments of organizational mechanisms, and will then focus on unsupervised, supervised and reinforcement learning. Another aim of this introductory review is the focus on relating different approaches in machine learning such as SVM and Bayesian networks, or reinforcement learning and temporal supervised learning. Some examples of relations to brain processing are included such as synaptic plasticity and models of the Basal Ganglia. I also provide Matlab examples for each of the three main learning paradigms with programs available at `www.cs.dal.ca/∼tt/repository/MLintro2012`.

1 Evolution, Development and Learning

Development and learning are both important ingredients for the success of natural organisms, and applying those concepts to artificial systems might hold the key to new breakthroughs in science and technology. This article is an introduction to machine learning with examples of its relation to neuroscientific findings. There has been much progress in this area, specifically by realizing the importance of representing uncertainties and the corresponding usefulness of a probabilistic framework.

1.1 Organizational mechanisms

Before focusing on the main learning paradigms that are dominating much of our recent thinking in machine learning, I would like to start by outlining briefly some

Thomas Trappenberg
Dalhousie University, Halifax Canada, e-mail: tt@cs.dal.ca

of my views on the intimate relations between organizational mechanisms discussed in this volume. It seems to me that there are at least three levels of organizational mechanisms that contribute to the success of living organisms, evolutionary mechanisms, developmental mechanisms and learning mechanisms. *Evolutionary mechanisms* seems to focus on the long-term search for suitable architectures. This search takes time; it takes usually many generations to establish small modifications that are beneficial for the survival of a species, and it takes even longer to branch off new species that can exploit niches in the environment. This mechanisms is adaptive in that it depends on the environment, the physical space and other organisms. A good principle organization and good choices of an organisms ultimately determine survival of the individuals and thereby to the species in general.

While evolution seems to work on a general architectural level of a population, a principle architecture has to be realized in specific individuals. This is where *development* comes into play. The genetic code can be used to grow a specific organisms from the master plan and the environmental conditions. Thus, this mechanisms is itself adaptive in that the environment can influence the specific encoding of the master plan. For example, the physique and metabolism of the socky salmon can change drastically when environmental conditions allow migration from the freshwater environment to the ocean, though the fish stays small and adapted to fresh water if prevented from migration or if food sources are sufficient in the river. Growing specific architectures in response to environmental conditions can give an organisms considerable advantages. Epigenetics, which is the study of how the environment can influence genetic decoding, is a fascinating new area in science.

Having grown a specific architecture, the resulting organisms can continue to respond to environmental conditions by learning about specific situations and by learning to take appropriate actions. This type of adaptation (learning) of a specific learning architecture can take several forms. For example, the learning can be supervised by other individuals such as the parents of an offspring to teach behavioural patterns that the parents finds advantageous. The organisms can also learn from more general environmental feedback by receiving reinforcing signals such as food or punishment. This article will focus mainly on such learning mechanisms.

The three different adaptive frameworks outline above are somewhat abstract at this level and it is important to give it more specific meaning with specific implementations. But this is also when the distinction between the different mechanisms can becomes a bit murky. For example, the development of receptive fields during the critical postnatal periods are certainly an important mechanisms on a developmental level, but we will discuss such mechanisms as special form of learning in this chapter. For the sake of this volume it is indeed useful to think about the learning mechanisms described in this section as the mechanisms that allow the fine tuning of the systems to specific environmental conditions as experienced by the specific individual during its lifetime, while other mechanisms discussed in this volume are aimed to develop better learning systems in the long term or to grow specific individuals in response to environmental conditions.

1.2 Generalization

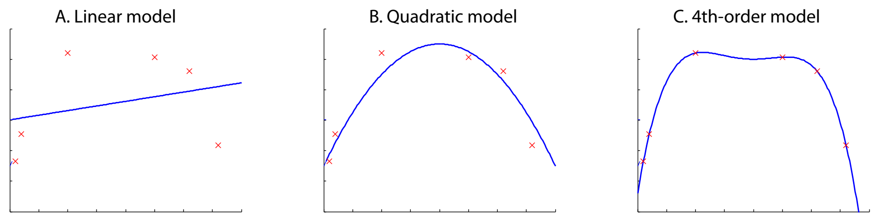

Fig. 1 Example of under- and overfitting.

An general goal of a learning system as described here is to predicting labels for future unseen data. The examples give during a learning phase are used to choose parameters of the model so that a specific realization of this model can make good predictions. The quality of generalization from training data depends crucially on the complexity of the model that is hypothesized to describe the data as well as the number of training data. This is illustrated in the left graph of Fig.1. Let's think about describing the six data points shown there with a linear model. Its regression curve is shown in the left graph, while the other two graphs show the regression with a quadratic model and with a model where the highest exponent is 4. Certainly, the linear model seem too low dimensional since the data points deviate systematically with the data points in the middle laying above the curve and the data points at the ends laying below the curve. Such a systematic *bias* is a good indication that the model complexity is too low. The curve on the right fits the data perfectly. We can always achieve a perfect fit of a finite number of training data if the number of free parameters (one for each order of the polynomial in this example) approaches the number of training points. But this could be *overfitting* the data. To evaluate if we are overfitting we need additional validation examples. In case of overfitting the *variance* of this validation error grows with increasing model complexity.

The *bias-variance tradeoff* that we just discussed is summarized in the left graph of Fig. 2. Many advances in machine learning have been made by addressing ways to choose good models. While the bias-variance tradeoff is now well appreciated in the machine learning community, many methods are based on general learning machines that have a large number of parameters. For such machines it is now common that a meta-learning method address the bias-variance tradeoff to find the points of minimal generalization error. However, an important question we need to consider is if the models take all necessary factors into account. How about including new features not previously considered such as the inclusion of a temporal domain? I believe that genetic and developmental mechanisms can address these issues by exploring different architectures or structures. However, only exploring architectural varieties of models is also not sufficient to find the best possible generalization per-

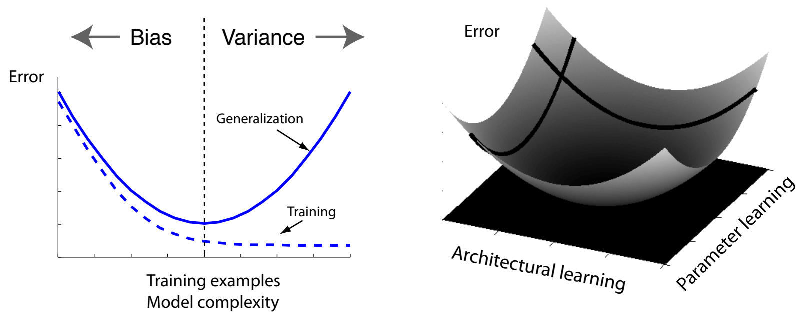

Fig. 2 Bias-variance tradeoff and explorative learning.

formance, and I believe that parameter optimization in the sense of supervised learning discussed below must accompany the architectural exploration. Several of the contributions in this volume are good examples of this.

1.3 Learning with uncertainties

In the following sections I give a modern view of learning theories that includes unsupervised, supervised and reinforcement learning. I will start with unsupervised learning since this is likely less known and relates closer to some developmental aspects of an organisms. I will then briefly review supervised learning in a probabilistic framework. Finally I will discuss reinforcement learning as an important generalization of supervised learning. I will also discuss some relations of these learning theories with biological analogies. In particular, I will outline the relations of unsupervised learning with the development of filters in early sensory cortices, discuss synaptic plasticity as the physical basis of learning, and I outline research that relates to the Basal Ganglia which has intriguing analogies with reinforcement learning theories.

Machine Learning has recently revolutionized computer applications such as autonomously driving cars or searching for information. Two major ingredients have been contributing to the recent success. The first is building into the system the ability to adapt to unforeseen events. In other words, we must build machines that learn since the traditional method of encoding appropriate responses to all future situations is impossible. Like humans, machines should not be static entities that only blindly follow orders which might be outdated by the time real situations are encountered. Although learning machines have been studied for at least half a century, often inspired by human capabilities, the field has matured considerably in recent years through more rigorous formulations of learning machines and the realization of the importance of predicting previously unseen events rather than only memorizing previous events. Machine learning is now a well established discipline within artificial intelligence.

The second ingredient for the recent breakthroughs is the acknowledgment that there are uncertainties in the world. Rather than only following the most likely explanations for a given situation, keeping an open mind and considering other possible explanations has proven to be essential in systems that have to work in a real world environment in contrast to controlled lab environment. The language of describing uncertainty, that of probability theory, has proven to be elegant and tremendously simplifies arguing in such worlds. This chapter is dedicated to an introduction to the probabilistic formulation of machine learning.

It was important to me to include supervised, unsupervised and reinforcement learning in this review in the form that I think matches advanced treatments in a machine learning course of machine learning. While there are now many good publications which focus on specific approaches in machine learning (such as Kernel methods or Bayesian models), my aim is to relate and contrast several popular learning approaches. While it is common to start with supervised learning, I opted for starting with a discussion of unsupervised learning as this logically precedes supervised learning and is generally less familiar compared to supervised learning.

1.4 Predictive learning

Finally, I include some examples of how machine learning topics might be related to neuroscientific issues. As already mentioned, the goal of learning as described here is anticipation or prediction. I believe that this is also the general goal of the brain, making good predictions to aid survival and evolutionary advantages. A possible architecture of good learning systems is outlined in Fig.3. An agent has to interact with the environment from which it learns and receives reward. This interaction has two parts, sensation and action. The state of the environment is conveyed by sensations that are caused by specific situations in the environment. A comprehension of these sensation requires hierarchical processing in deep learning systems. The hierarchical processes are bidirectional so that the same structures can also be used to generate expectation which should ultimately generate good actions. These actions have to be guided by a value system that need to learn itself from the environment. This chapter reviews the components of such learning systems.

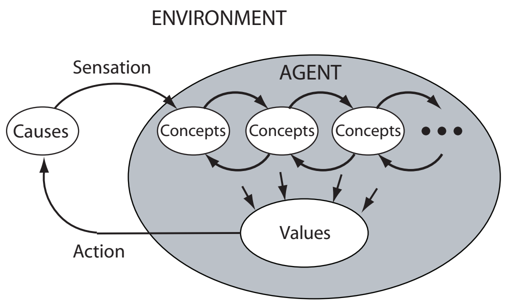

Fig. 3 The anticipating brain with deep believe networks and a value system.

2 Unsupervised Learning

2.1 Representations

An important requirement for natural or artificial agents is to decide on an appropriate course of action given its specific circumstances of an encountered environment. We can treat the environmental circumstances as cues given to the agent. These cues are communicated by sensor that specify values of certain features. Let's represent these *feature values* as vector $\mathbf{x}$. The goal of the agent is then to calculate an appropriate responses

$$y = f(\mathbf{x}). \tag{1}$$

In this review we use a probabilistic framework to address uncertainties. The corresponding statement of the deterministic function approximation of equation (1) is then to find a probability density function

$$p(y|\mathbf{x}). \tag{2}$$

A common example is object recognition where the feature values might be RGB values of pixels in a digital image and the desired response might be the identification of a person in this image. A learning machines for such a task is a model that is given examples with explicit features vectors $\mathbf{x}$ and desired *labels* y. Learning in this circumstance is mainly about adjusting the model's parameters from the given examples. A trained machine should be able to generalize by predicting labels of previously unseen feature vectors. Since this type of learning is based on specific training examples with give labels, this type of learning is called *supervised*. We will discuss specific algorithms of supervised learning and corresponding models in the next section. Here we start with discussing unsupervised learning since this is a more fundamental tasks that precedes supervised learning.

As already stated, the aim of learning is to find a mapping function $y = f(\mathbf{x})$ or probability density function $p(y|f(\mathbf{x}))$. An important insight that we explore in this sections is that finding such relations is much easier if the representation of the feature vector is chosen carefully. For example, it is actually very challenging to use raw pixel values to infer the content of a digital photo. In contrast, if we have given a useful descriptions of faces, such as the distance between eyes and other landmark features, the colour of hair, and the length of the nose etc, it is much easier to classify photographs to specific target faces. Finding a useful representation of a problem is often key for successful applications. When we use learning techniques for this task we talk about *representational learning*. Representational learning is often exploiting statistical characteristics of the environment without the need of labeled training examples. This is therefore an important area of *unsupervised learning*.

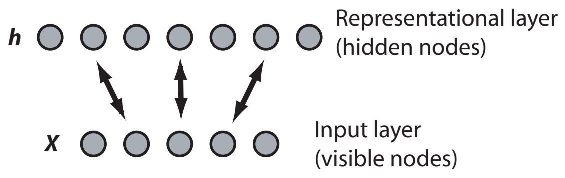

Fig. 4 Restricted Boltzmann machine which is a probabilistic two layer network with bidirectional symmetric connections between the input layer and the representational (hidden) layer.

Representational learning can be viewed itself as a mapping problem, such as the mapping from raw pixel values to more direct features of a face. This is illustrated in Fig.4 where the raw input feature vector, **x**, is represented by a layer of nodes at the bottom. Let's call this layer the *input layer*. The feature vector for higher order representations, **h**, is represented as nodes in the upper layer of this network. Let's call this the *representational layer* or *hidden layer*. The connections between the nodes represent the desired transformation between input layer and hidden layer. In line with our probabilistic framework, each node represents a random variable. The main idea of the principle that we will employ to find useful representations is that this representations should be useful in reconstructing the input.

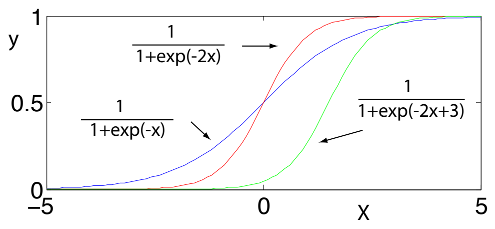

Fig. 5 Logistic function with different slopes and offsets.

Before we discuss different variations of hidden representations, let us make the functions of the model more concrete. Specifically, let us consider mainly binary random variables for illustration purposes. Given the values of the inputs, we choose to calculate the value of the hidden nodes with index i, or more precisely the probability of having a certain value, with a logistic function shown in Fig.5,

$$p(h_i = 1|\mathbf{x}) = \frac{1}{1+e^{-\frac{1}{T}(\mathbf{w}_i\mathbf{x}+b_i)}}, \tag{3}$$

where T is a temperature parameter controlling the steepness of the curve, $\mathbf{w}$ are the weight values of the connections between input and hidden layer, and b_i^{h} is the offset of the logistic function, also called bias of the hidden node. In this model, which is called a restricted Boltzmann machine [4], there are no connections between hidden nodes so that the hidden nodes represent random variables that are conditionally independent when the inputs are observed. In other words, the joined density function with fixed inputs factorizes,

$$p(\mathbf{h}|\mathbf{x}) = \prod_i \frac{1}{1+e^{-\frac{1}{T}\sum_j w_{ij}x_j+b_i^{\mathrm{h}}}}. \tag{4}$$

The connections in this model are bidirectional, and such a model represents therefore a symmetric Bayesian network discussed further below. The state of the input nodes can be generated by hidden activations like

$$p(\mathbf{x}|\mathbf{h}) = \prod_i \frac{1}{1+e^{-\frac{1}{T}\sum_j w_{ij}h_j+b_i^{\mathrm{v}}}}, \tag{5}$$

where b_i^{v} are the biases for each visible (input) node.

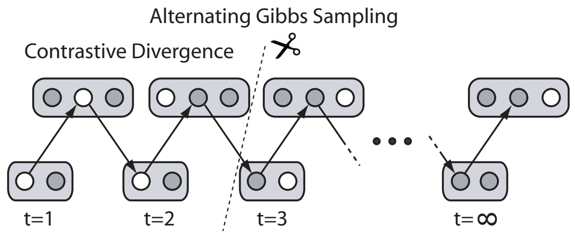

Fig. 6 Alternating Gibbs sampling.

The remaining question is how to choose parameters, specifically the weights and biases of the model? Since our aim is to reconstruct the world, we can formulate this in a probabilistic framework by minimizing the distance between the world distribution (the density function of visible nodes when set by unlabeled examples from the environment) and the generated model of the world when sampled from hidden activities. The difference between distributions is often measured with the Kullbach-Leibler divergence, and minimizing this objective function with a gradient method leads to a Hebbian-type learning rule

$$\Delta w_{ij} = \eta \frac{\partial l}{\partial w_{ij}} = \eta \frac{1}{2T} \left(\langle s_i s_j \rangle_{\text{clamped}} - \langle s_i s_j \rangle_{\text{free}} \right). \tag{6}$$

The angular brackets $\langle . \rangle$ denote thermal averages, either in the clamped mode where the inputs are fixed or in the free running mode where the input nodes activity is determined by the hidden nodes. unfortunately, this learning rule suffers in practice from the long time it takes to produce thermal average. However, it turns out that learning still works for a few steps in Gibbs sampling as illustrated in Fig.6. This learning rule is called contrastive divergence [1].

An example of a basic restricted Boltzmann machine is given in Table 1. This network is used to learn digitized letters of the alphabet that are provided in file `pattern1.txt` at `www.cs.dal.ca/~tt/repository/MLintro2012` together with the other programs of this article. This RBM has $nh = 100$ hidden nodes and is trained for $nepochs = 150$ epoch, where one epoch consists of presenting all images once. The network is trained with contrastive divergence in the next block of code. The training curve, which shows the average error of recall of pattern, is shown on the left in Fig.7. After training, 20% of the bits of the training patterns are flipped and presented as input to the network. Then plots the patterns after repeated reconstructions as shown on the right in Fig.7. Only the first 5 letters are shown here, but this number can be increased to inspect more letters.

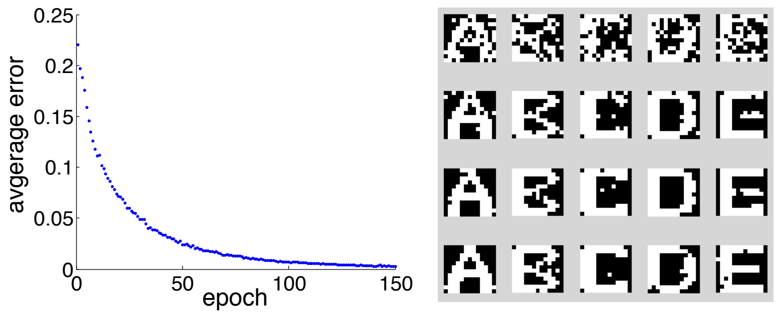

Fig. 7 Output of the example program for a restricted Boltzmann machine. The learning curve on the left shows the development of the average reconstruction error, and the reconstructions of noisy patterns after training are shown on the right.

2.2 *Sparse and topographic representations*

In the previous section we reviewed a basic probabilistic network that implements representational learning based on reconstructions of inputs. There are many other unsupervised algorithms that can do representational learning such as non-probabilistic recurrent networks. Also, many other representational learning algo-

Table 1 Basic restricted Boltzmann machine to learn letter patterns

```
clear; nh=100;  nepochs=150;  lrate=0.01;
%load data from text file and rearrange into matrix
load pattern1.txt;
letters = permute( reshape( pattern1, [12 26 13]), [1 3 2]);

%%train rbm for nepochs presentations of the 26 letters
input = reshape(letters,[12*13 26])
vb =zeros(12*13,1);  hb =zeros(nh,1);  w =.1*randn(nh,12*13);

figure; hold on;
xlabel 'epoch'; ylabel 'error'; xlim([0 nepochs]);
for epoch=1:nepochs;
  err=0;
  for i=1:26
    %Sample hidden units given input, then reconstruct.
    v = input(:,i);
    h = 1./(1 + exp(-(w *v + hb))); %sigmoidal activation
    hs= h > rand(nh,1);             %probabilistic sampling
    vr= 1./(1 + exp(-(w'*hs+ vb))); %input reconstruction
    hr= 1./(1 + exp(-(w *vr+ hb))); %hidden reconstruction

    %Contrastive Divergence rule: dw ~ h*v - hr*vr
    dw  = lrate*(h*v'-hr*vr');  w = w +dw;
    dvb = lrate*( v  -  vr  );  vb= vb+dvb;
    dhb = lrate*( h  -  hr  );  hb= hb+dhb;
    err = err   + sum( (v-vr).^2 );   %reconstruction error
  end
  plot( epoch, err/(12*13*26), '.');  drawnow;%figure output
end

%%plot reconstructions of noisy letters
r = randomFlipMatrix(round(.2*12*13)); %(20% of bits flipped)
noisy_letters = abs(letters - reshape(r,[12 13 26]));
recon = reshape(noisy_letters, 12*13, 26); %put data in matrix
recon=recon(:,1:5);                        %only plot first 10
figure; set(gcf,'Position',get(0,'screensize'));

for i=0:3
  for j=1:5
    subplot(3+1, 5, i*5 + j);
    imagesc( reshape(recon(:,j),[12 13]) );  %plot
    colormap gray; axis off; axis image;

    h = 1./(1 + exp(-(w *recon(:,j) + hb))); %compute hidden
    hs= h > rand(nh,1);                      %sample hidden
    recon(:,j) = 1./(1 + exp(-(w'*hs + vb)));%compute visible
    recon(:,j) = recon(:,j) > rand(12*13,1); %sample visible
  end
end

function r=randomFlipMatrix(n);
% returns matrix with components 1 at n random positions
r=zeros(156,26);
for i=1:26
    x=randperm(156);
    r(x(1:n),i)=1;
end
```

rithm are known from signal processing such as Fourier transformation, wavelet analysis, or independent component analysis (ICA). Indeed, most advanced signal processing include steps to re-representing or decompose a signal into basis functions. For example, the Fourier transformation decomposes a signal into sine waves with different amplitudes and phases. The individual sine waves form a dictionary and the original signal is represented with the coefficient for each of these basis functions. An example is shown in Fig.8. The signal in the upper left is made out of three sine waves, as revealed by the power spectrum on the right.

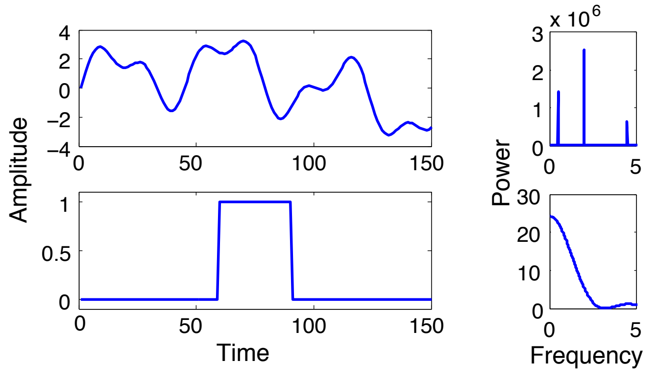

Fig. 8 Decomposition of signals into sine waves. The example signals are shown on the left side, and the corresponding description of the power spectrum on the right. The power spectrum shows the square of the amplitude for each contributing sine wave with specified frequency.

The Fourier transformation has been very useful in describing periodic signals, but one problem with this representation is that an infinite number of basis functions is need to represent a signal that is localized in time. An example of a square signal localized in time is shown in the lower left panel of Fig.8 together with its power spectrum on the right. The power spectrum plots the absolute of the amplitudes for different frequencies of the Fourier transform. In the case of the time-localized signal, the power spectrum shows that a continuous number of frequencies are necessary to accurately represent the original signal. A better choice for such formulations would be basis functions that are localized in time. An example of such transformations are wavelet transforms [2] or the Huang-Hilbert transform [3]. The usefulness of a specific transformation depends of course on the nature of the signals. Periodic signals with few frequency components, such as the rhythm of the heart or yearly fluctuations of natural events, are well represented by Fourier transforms, while signals with localized features, such as objects in a visual scene, are often well represented with wavelets. The main reason for calling a representation useful is that the original signal can be represented with only a small number of basis functions, or with other words, when only a small number of coefficients have significant large values. Thus, even if the dictionary might be large, each example of a signal of the specific environment can be represented with a small number of components. Such representations are called *sparse*.

The major question is then how to find good (sparse) representations for specific environments. One solution within the learning domain is to learn representations by unsupervised learning as demonstrated above with the example of a Boltzmann machine. To learn sparse representations we now add additional constrains that force the learning of specific basis functions. In order to do this we can keep track of the mean activation of the hidden nodes,

$$q_j(t) = (1-\lambda)q_j(t-1) + \lambda h_j(t), \tag{7}$$

where the parameter λ determines the averaging window. We then add the constraint of minimize the difference between the *desired sparseness* ρ and the *actual sparseness* q to the learning rule,

$$\Delta w_{ij} \propto v_i(h_j + \rho - q_j) - v_i^r h_j^r. \tag{8}$$

This works well in practice and has the added advantage of preventing *dead nodes* [5]. The importance of sparse representations in the visual system has long been pointed out by Horace Barlow [6], and one of the best and probably first examples that demonstrate such mechanisms was give by his student Peter Foldiak [7] (see also [8]). Another very influential paper was that by Olshausen and Field [9] who demonstrated that sparseness constrains are essential in learning basis functions that resemble receptive fields in the primary visual cortex, and similar concepts should also hold for higher order representations in deep believe networks [10]. It is now argued that such unsupervised mechanisms resemble receptive fields of simple cells.

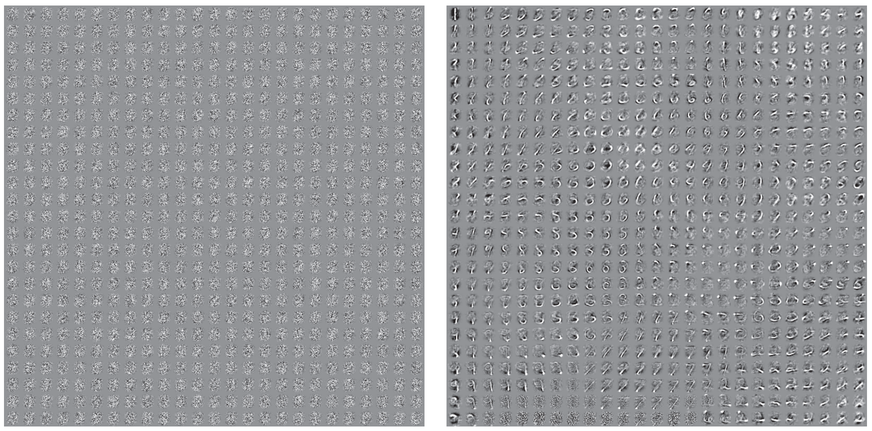

Fig. 9 Examples of learned receptive fields of a RBM without (left) and with (right) sparse and topographic constrains.

In addition to the typical from of receptive fields, many brain areas show some *topographic organizations* in that neurons with adjacent features of receptive fields are located in adjacent tissue. An example of unsupervised topographic representations is the self-organizing map (SOM) [11]. Topographic self-organization can be triggered by lateral interactions with local facilitation and distant competition as can be implemented with pairwise local excitation and distant inhibition between neurons. Such interactions also promote sparse representations. My student Paul Hollensen, together with my collaborator Pitoyo Hartonon and me, did therefore propose to lateral interactions within the hidden layer [12] such as,

$$p(\hat{h}_j|\mathbf{v}) = \Sigma_k \mathscr{N}_{j,k} p(h_k|\mathbf{v}) \tag{9}$$

where $\mathscr{N}_{j,k}$ is a Kernel such as a shifted Gaussian or a Mexican-hat function centered on hidden node j. For binary hidden units the natural measure of the difference in distributions is the cross entropy, for which the derivative with respect to the weights is simply $(\hat{h}_j - h_j) \cdot \mathbf{v}$. Combining this with the CD update yields

$$\Delta w_{ij} \propto v_i h_j - v_i^r h_j^r + (\hat{h}_j - h_j) v_i = v_i \hat{h}_j - v_i^r h_j^r. \tag{10}$$

Examples of receptive field learned with (right) and without (left) sparse topographic learning are shown in Fig.9.

3 Supervised Learning

3.1 Regression

The representational learning is about learning a mapping function that transforms a signal (input vector) to a new signal (hidden vector),

$$f_r : \mathbf{x} \to \mathbf{h} \quad \text{(given unlabeled examples and constrains).} \tag{11}$$

The unsupervised learning of this mapping function typically exploiting statistical regularities in the signals on depends therefore on the nature of the input signals. This learning is also guided by principles such as good reconstruction abilities, sparseness and topography. Supervised learning is about learning an unknown mapping function from labeled examples,

$$f_y : \mathbf{h} \to \mathbf{y} \quad \text{(given labeled examples).} \tag{12}$$

We have indicated in the formula above that supervised learning takes the hidden representation of examples, **h** to map then to a desired output vector **y**. This assumes that representational learning is somewhat completed during a developmental learning phase, which is then followed by supervised learning with a teacher that supplies desired labels (output values) for given examples. It may be argued that in natural learning systems these learning phases are not as strictly separated as discussed here, but for the purpose of this tutorial it is useful to make this distinctions of principle learning components.

By discussing now strictly supervised learning, let us follow the common nomenclature in represent input values as $\mathbf{x}$ and output values as $\mathbf{y}$. In supervised learning we consider trainnig data that consist of example inputs and corresponding labels, that is, pairs of values $(\mathbf{x}^{(i)}\mathbf{y}^{(i)})$, where the index $i = 1,...,m$ labels each of m training example. An example is shown in Fig.10 that partially lists and plots all running records of 30 employees who were regular members of a company's health club [13]. Specifically, the data show the relation between the weight of the persons and their time in a one-mile run.

Weight (in pounds)	Time of one-mile run (in seconds)
217	481
141	292
152	338
153	357
180	396
.	
.	
.	
245	469
141	252
177	338

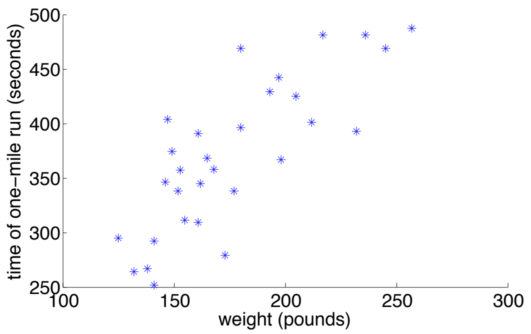

Fig. 10 Health data.

Looking at the plot reveals that there seems to be a systematic relation between the weights and running times, with a trend that heavier persons tend to be slower in the runs, although this is not true for all individuals. Moreover, the trend seems linear. This hypothesis can be quantified as a parameterized function,

$$h(x;\theta) = \theta_0 + \theta_1 x. \tag{13}$$

This notation means that the hypothesis h is a function of the quantity x, and the hypothesis includes all possible straight lines, where each line can have a different offset θ_0 (intercept with the y-axis), and slope θ_1. We typically collect parameters in a *parameter vector* θ. We only considered one input feature x above, but we can easily generalize this to higher dimensional problems where more input *attributes* are given. For example, there might be the amount of exercise each week that might impact the results of running times. If we make the hypothesis that this additional variable has also a linear influence on the running time, independently of the other attribute, we can write the hypothesis as

$$h(\mathbf{x};\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2. \tag{14}$$

A useful trick to enable a compact notation in higher dimensions with n attributes is to introduce $x_0 = 1$. We can then write the linear equations as

$$h(\mathbf{x};\theta) = \theta_0 x_0 + ... + \theta_n x_n = \sum_i \theta_i x_i = \theta^T \mathbf{x}. \tag{15}$$

The vector θ^T is the transpose of the vector θ.

At this point it would be common to fit the unknown parameters θ with methods such as least mean square regression (LMS). However, I would like to frame this problem right away in a more modern probabilistic framework. The data already show that the relations between the weight and running time is not strictly linear, and the main question is how we should interpret the differences. We could introduce a more complicated nonlinear hypothesis that tries to fit the deviations from the linear hypothesis. Such a hypothesis would be good news since increasing your weight from 180 pounds to 200 pounds would predict that people can run faster. Thus, instead of making the hypothesis function more complex, we should consider other possible sources that influence these data. One is certainly that the ability to run does not only depend on the weight of a person but also on other physiological factors. However, these data do not include information of such other factors, and the best we can other than collecting more information is to treat these deviations as *uncertainties*. There are many possible source of uncertainties such as limitations of measurements from either time constrains, laziness or sensor limitations, or true noise in the data. At this point it is not important where these uncertainties originate but only that we acknowledge the uncertain nature of data.

To model the uncertainties in these data we look at the deviations from the mean. Fig.11 shows a histogram of the differences between the actual data and the hypothesised regression line. This look a bit Gaussian, which is a frequent finding in data

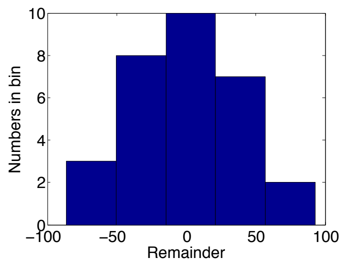

Fig. 11 Histogram of the difference between the data points and the fitted hypothesis, $(y_i - \theta_1 - \theta_2 x_i)$.

though not necessarily the only possible. With this additional conjecture, we should revise our hypothesis. More precisely, we acknowledge that the data are noisy and that we can only give a probability of finding certain values. Specifically, we assume here that the data follow a certain trend $h(\mathbf{x};\theta)$ with *additive noise*, η,

$$p(y|x;\theta) = h(\mathbf{x};\theta) + \eta, \tag{16}$$

where the random variable η is Gaussian distributed in the example above,

$$p(\eta) = \aleph(\mu, \sigma) \tag{17}$$

We can then also write the probabilistic hypothesis in the above example as a Gaussian model with a mean that depends on the variable x,

$$p(y|x;\theta) = \aleph(\mu = h(x), \sigma) \tag{18}$$

$$= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{-(y-\theta^T\mathbf{x})^2}{2\sigma^2}\right) \tag{19}$$

This functions specifies the probability of values for y, given an input x and the parameters θ. We have here treated here the variance *sigma*^{2} as given, although this can be part of the model parameters that need to be estimated.

Specifying a model with a density function is an important step in modern modelling and machine learning. In this type of thinking, we treat data from the outset as fundamentally stochastic, that is, data can be different even in situations that we deem identical. This randomness may come from *irreducible indeterminacy*, but might also represent *epistemological limitations* such as the lack of knowledge of hidden processes or limitations in observing states directly. The language of probability theory has helped to make large progress the machine learning area.

While we have made a parameterized hypothesis underlying the nature of data, we need to estimate values for the parameters to make real predictions. We therefore consider again the examples for the input-output pairs, our training set $\{(x^{(i)}, y^{(i)}); i = 1...m\}$. The important principle that we will now follow is to choose the parameters so that the examples we have are most likely under the model. This is called *maximum likelihood estimation*. To formalize this principle, we need to think about how to combine probabilities for several observations. If the observations are independent, then the joined probability of several observations is the product of the individual probabilities,

$$p(y_1, y_2, ..., y_m | x_1, x_2, ..., x_m; \theta) = \Pi_i^m p(y_i | x_i; \theta). \tag{20}$$

Note that y_i are still random variables in the above formula. We now use our training examples as specific observations for each of these random variables, and introduce the *Likelihood function*

$$L(\theta) = \Pi_i^m p(\theta; y^{(i)}, x^{(i)}). \tag{21}$$

The p on the right hand side is now not a density function, but it is a regular function (with the same form as our parameterized hypothesis) of the parameters θ for the given values $y^{(i)}$ and $x^{(i)}$. Instead of evaluating this large product, it is common to use the logarithm of the likelihood function, so that we can use the sum over the training examples,

$$l(\theta) = \log L(\theta) = \sum_i^m \log(p(\theta; y^{(i)}, x^{(i)})). \tag{22}$$

Since the log function strictly monotonly rising, the maximum of L is also the maximum of l. The maximum (log-)likelihood can thus be calculated from the examples as

$$\theta^{\text{MLE}} = arg \max_\theta l(\theta). \tag{23}$$

We might be able to calculate this analytically or use a search algorithms to find a minimum from this function.

Let us apply this to the regression of a linear function as discussed above. The log-likelihood function for this example is

$$\begin{aligned} l(\theta) &= \log \Pi_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{-(y^{(i)} - \theta^T \mathbf{x}^{(\mathbf{i})})^2}{2\sigma^2}\right) &(24)\\ &= \sum_{i=1}^m \left(\log \frac{1}{\sqrt{2\pi}\sigma} - \frac{-(y^{(i)} - \theta^T \mathbf{x}^{(\mathbf{i})})^2}{2\sigma^2}\right) &(25)\\ &= -\frac{m}{2}\log 2\pi\sigma - \sum_{i=1}^m \frac{-(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2}{2\sigma^2}. &(26)\end{aligned}$$

Thus, the log was chosen so that we can use the sum in the estimate instead of dealing with big numbers based on the product of the examples. Since the first term in the expression (26), $-\frac{m}{2}\log 2\pi\sigma$, is independent of θ, and since we considered here a model with a constant σ^2 for the variance of the data, maximizing the log-likelihood function is equivalent to minimizing a quadratic error term

$$E = \frac{1}{2}(y - h((x; (\theta))^2 \Longleftrightarrow p(y|\mathbf{x}; \theta) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{((y - h((x; \theta)^2}{2}). \tag{27}$$

Thus, maximum likelihood estimation of a Gaussian data correspond to minimizing a quadratic cost function as was commonly used in LMS regression. LMS regression is thus well motivated for Gaussian data, but our deviation also show that data with non-Gaussian noise should be fitted with different cost functions. For example, a *polynomial error function* correspond more generally to a density model of the form

$$E = \frac{1}{p}||y - h((x; (\theta)||^p \Longleftrightarrow p(y|\mathbf{x}; \theta) = \frac{1}{2\Gamma(1/p)} \exp(-||y - h((x; \theta||^p). \tag{28}$$

Later we will mention the *ε-insensitive error function*, where errors less than a constant ε do not contribute to the error measure, only errors above this value,

$$E = ||y - h((x; (\theta)||_\varepsilon \Longleftrightarrow p(y|\mathbf{x}; \theta) = \frac{p}{2(1-\varepsilon)} \exp(-||y - h((x; \theta||_\varepsilon). \tag{29}$$

Since we already acknowledged that we do expect that data are noisy, it is somewhat logical to not count some deviations form the expectation as errors. It also turns out that this error function is much more robust than other error functions.

3.2 Classification as logistic regression

We have grounded supervised learning in probabilistic function regression and maximum likelihood estimation. An important special case of supervised learning is *classification*, The simplest example is that of binary classification which are data that have only two possible labels such as $y = (0,1)$. For example, let us consider a one dimensional case where data tend to fall into one class when the feature value is below a threshold θ, or into the other class when above. The most difficult situation for such classification is around the threshold value since small changes in this value might trigger one versus the other class. It is then appropriate to make a hypothesis for the probability of $p(y=1|x)$ which is small for x ($x << \theta$), around 0.5 for $x \approx \theta$, and approaching one for large x ($x >> \theta$).

More formally, let us consider a random number which takes the value of 1 with probability ϕ and the value 0 with probability $1-\phi$ (the probability of being either of the two choices has to be 1.) Such a random variable is called Bernoulli distributed. Tossing a coin is a good example of a process that generates a Bernoulli random variable and we can use maximum likelihood estimation to estimate the parameter ϕ from such trials. That is, let us consider m tosses in which h heads have been found. The log-likelihood of having h heads ($y=1$) and $1-h$ tails ($y=0$) is

$$l(\phi) = \log(\phi^h (1-\phi)^{m-h}) \quad (30)$$

$$= h\log(\phi) + (m-h)\log(1-\phi). \quad (31)$$

To find the maximum with respect to ϕ we set the derivative of l to zero,

$$\frac{\mathrm{d}l}{\mathrm{d}\phi} = \frac{h}{\phi} - \frac{m-h}{1-\phi} \quad (32)$$

$$= \frac{h}{\phi} - \frac{m-h}{1-\phi} \quad (33)$$

$$= 0 \quad (34)$$

$$\rightarrow \quad \phi = \frac{h}{m} \quad (35)$$

As you might have expected, the maximum likelihood estimate of the parameter ϕ is the fraction of heads in m trials.

Now let us discuss the case when the probability of observing a head or tail, the parameter ϕ, depends on an attribute x, as usual in a stochastic (noisy) way. An example is illustrated in Fig.12 with 100 examples plotted with star symbols. The data suggest that it is far more likely that the class is $y=0$ for small values of x and that the class is $y = 1$ for large values of x, and the probabilities are more similar in-between. We put forward the hypothesis that the transition between the low and high probability region is smooth and qualify this hypothesis as parameterized density function known as a *logistic* (sigmoidal) function

$$p(y = 1) = \frac{1}{1 + \exp(-\theta^T \mathbf{x})}. \tag{36}$$

As before, we can then treat this density function as function of the parameters θ for the given data values (likelihood function), and use maximum likelihood estimation to estimate values for the parameters so that the data are most likely.

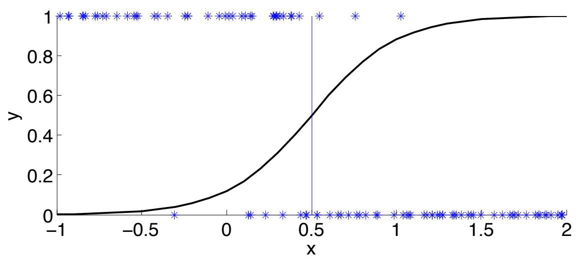

Fig. 12 Binary random numbers (stars) drawn from the density $p(y = 1) = \frac{1}{1+\exp(-\theta_1 x - \theta_0)}$ (solid line) with offset $\theta_0 = 2$ and slope $\theta_1 = 4$.

How can we use the knowledge (estimate) of the density function to do classification? The obvious choice is to predict the class with the higher probability, given the input attribute. This *bayesian decision point*, x_d, or *dividing hyperplane* in higher dimensions, is give by

$$p(y = 1|x_d) = p(y = 0|x_d) = 0.5 \rightarrow x_d \theta^T \mathbf{x_d} = 0. \tag{37}$$

We have here considered binary classification with linear decision boundaries as logistic regression, and we can also generalize this method to problems with non-linear decision boundaries by considering hypothesis with different functional forms of the decision boundary. However, coming up with specific functions for boundaries is often difficult in practice, and we will discuss much more practical methods for binary classification later in this course.

3.3 Multivariate generative models and probabilistic reasoning

We have so far only considered very simple hypothesis appropriate for the low dimensional data given in the above examples. An important issues that has to be considered in machine learning is that of generalizing to more complex non-linear data in high dimension, that is, when many factors interact in a complicated way. This topic is probably one of the most important when applying ML to real world data. This section discusses a useful way of formulating more complicated stochastic models with causal relations and how to use such models to argue (inference). Parameters of such models must often be learned with supervised techniques such as maximum likelihood estimation.

Let us consider high dimensional data and the corresponding supervised learning. In the probabilistic framework, this means making a hypothesis for joined density function of the problem,

$$p(y, \mathbf{x}) = p(y, x_1, x_2, ...|\theta). \qquad (38)$$

With this joined density function we could argue about every possible situation in the environment. For example, we could again ask for classification or object recognition by calculating the conditional density function

$$p(y|\mathbf{x}) = p(y|x_1, x_2, ...; \theta). \qquad (39)$$

Of course, the general joined density function and even this conditional density function for high dimensional problems have typically many free parameters that we need to estimate. It is then useful to make more careful assumptions of causal relations that restrict the density functions. The object recognition formulation above is sometimes called a *discriminative approach* to object recognition because it tries to discriminate labels give the feature values. Another approach is to consider modelling the inverse (we drop in the following the parameter vector in the notation)

$$p(\mathbf{x}|y) = p(x_1, x_2, ...|y). \qquad (40)$$

This is called a *generative model* as it can generate examples from a class give a label. To use generative models in classification or object recognition we can use Bayes' rule to calculate a discriminative model. That is, we use *class priors* (the relative frequencies of the classes) to calculate the probability that an item with features **x** belong to a class *y*,

$$p(y|\mathbf{x}; \theta) = \frac{p(\mathbf{x}|y; \theta)p(y)}{p(x)}. \qquad (41)$$

While using generative models for classification seem to be much more elaborate, there are several reasons that make generative models attractive for machine learning. For example, in many cases features might be conditionally independent given a label, that is

$$p(x_1, x_2, ...|y) = p(x_1|y) * p(x_2|y) * .... \qquad (42)$$

Even if this does not hold strictly, this *naive Bayes assumption* it is often useful and drastically reduces the number of parameters that have to be estimates. This can be seen from factorizing the full joined density function with the chain rule

$$p(x_1,x_2,...,x_n|y) = p(x_n|y,x_1,...x_{n-1})p(x_1,...,x_{n-1}|y) \tag{43}$$
$$= p(x_n|y,x_1,...,x_{n-1}) * ... * p(x_2|y,x_1) * p(x_1|y) \tag{44}$$
$$= \prod_{i=1}^{n} p(x_i|y,x_{i-1},...x_1). \tag{45}$$

But what if the naive Bayes assumption is not appropriate? Then we need to build more elaborate models. Building and using such models have been greatly simplified with *graphical methods* to build *casual models* that specify the conditional dependencies between random variables [14]. A well known example of one of the inventors of graphical models, Judea Pearl, is shown in Fig.13. In such graphical models, the nodes represent random variables, and the links between them represent causal relations with conditional probabilities. In this case there are arrows on the links. This is thus an example of a *directed acyclic graph* (DAG). The RBM discussed above is a example of an undirected Bayesian network.

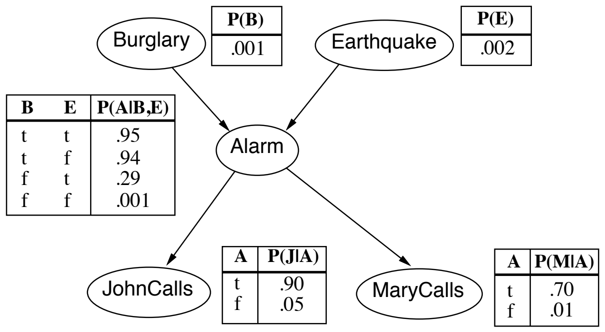

Fig. 13 Example of causal model a two-dimensional probability density function (pdf) and some examples of marginal pdfs.

In this specific example, each of the five nodes stands for a random binary variable (Burglary B={yes,no}, Earthquake E={yes,no}, Alarm A={yes,no}, JohnCalls J={yes,no}, MaryCalls M={yes,no}). The figure also include *conditional probability tables (CPTs)* that specify the conditional probabilities represented by the links between the nodes. The joined distribution of the five variables can be factories in various ways following the chain rule mentioned before (equations 43), for example as

$$p(B,E,A,J,M) = P(B|E,A,J,M)P(E|A,J,M)P(A|J,M)P(J|M)P(M) \tag{46}$$

However, the causal model represents a specific factorization of the joined probability functions, namely

$$p(B,E,A,J,M) = P(B)P(E)P(A|B,E)P(J|A)P(M|A), \tag{47}$$

which is much easier to handle. If we do not know the conditional probability functions, we need to run many more experiments to estimate the various conditions $(2^4+2^3+2^2+2^1+2^0=31)$ instead of the reduced conditions in the causal model $(1+1+2^2+2+2=10)$. It is also easy to use the casual model to do inference (drawing conclusions), for specific questions. For example, say we want to know the probability that there was no earthquake or burglary when the alarm rings and both John and Mary call. This is given by

$$\begin{aligned}
&P(B=f,E=f,A=t,J=t,M=t)= \\
&\quad =P(B=f)P(E=f,)P(A=t|B=f,E=f)P(J=t|A=t)P(M=t|A=t) \\
&\quad =0.998*0.999*0.001*0.7*0.9 \\
&\quad =0.00062
\end{aligned}$$

Although we have a casual model where parents variables influence the outcome of child variables, we can also use a child evidence to infer some possible values of parent variables. For example, let us calculate the probability that the alarm rings given that John calls, $P(A=t|J=t)$. For this we should first calculate the probability that the alarm rings as we need this later. This is given by

$$\begin{aligned}
P(A=t) &= P(A=t|B=t,E=t)P(B=t)P(E=t)+... \\
&\quad P(A=t|B=t,E=f)P(B=t)P(E=f)+... \\
&\quad P(A=t|B=f,E=t)P(B=f)P(E=t)+... \\
&\quad P(A=t|B=f,E=f)P(B=f)P(E=f) \\
&= 0.95*0.001*0.002+0.94*0.001*0.998+... \\
&\quad 0.29*0.999*0.002+0.001*0.999*0.998 \\
&= 0.0025
\end{aligned}$$

We can then use Bayes' rule to calculate the required probability,

$$\begin{aligned}
P(A=t|J=t) &= \frac{P(J=t|A=t)P(A=t)}{P(J=t|A=t)P(A=t)+P(J=t|A=f)P(A=f)} \\
&= \frac{0.90.0025}{0.90.0025+0.050.9975} \\
&= 0.0434
\end{aligned}$$

We can similarly apply the rules of probability theory to calculate other quantities, but these calculations can get cumbersome with larger graphs. It is therefore useful to use numerical tools to perform such inference. For example, a useful Matlab toolbox for Bayesian networks can be downloaded at `http://code.google.com/p/bnt/`. The Matlab implementation of the model in Fig.13 is implemented with this toolbox in file `www.cs.dal.ca/∼tt/repository/MLintro2012/PearlBurglary.m`

I mentioned already the importance of learning about temporal sequences (anticipatory systems), and Bayesian Networks are easily extended to this domain. An im-

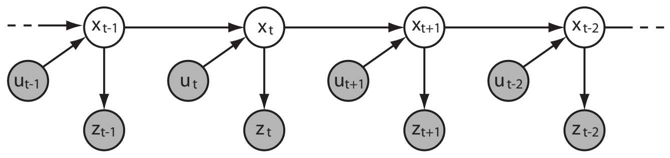

Fig. 14 A temporal Bayesian model called the Hidden Markov Model with hidden states x_t observations z_t and external influences u_t.

portant example of such a Dynamic Bayesian Network (DBN) is a Hidden Markov Model (HMM) as shown in Fig.14. In this model a state variable, x_t is not directly observed. This is therefore a *hidden* or *latent* random variable. The Markov condition in this model means that the states only depend on situations in the previous state, which can include external influences such described here as u_t. T typical example is a robot localization where we drive the robot with some motor command u_t and want to know the new state of the robot. We can use some knowledge about the influence of the motor command on the system to calculate a new expected location, and we can also combine this in a Baysian optima way with sensor measurement depicted as z_t. Such Bayesian models are essential in many robotics applications.

3.4 Non-linear regression and the bias-variance tradeoff

While graphical models are great to argue about situations (doing inference), the role of supervised learning is to determine the parameters of the model. We have only considered binary models where each Bernoulli variable is characterized by a single parameter ϕ. However, the density function can be much more complicated and introduce many more parameters. A major problem in practice is thus to have enough training examples with labels to restrict useful learning appropriately. This is one important reason for unsupervised learning as we have usually many unlabelled data that can be used to represent the problem appropriately. But we still need to understand the relations between free parameters and the number of training data.

We already discuss the bias-variance tradeoff in the first section. Finding the right function that describe nonlinear data is one of the most difficult tasks in modelling, and there is not a simple algorithm that can give us the answer. This is why more general learning machines, which we will discuss in the next section, are quite popular. To evaluate the generalization performance of a specific model it is useful to split the training data into a *training set*, which is used to estimate the parameters of the model, and a *validation set*, which is used to study the *generalization performance* on data that have not been used during training the model.

A important question is then how many data we should keep to validate versus train the model. If we use too many data for validation, than we might have too less data for accurate learning in the first place. On the other hand, if we have to few data for validation than this might not be very representative. In practice we are often using some *cross-validation* techniques to minimize the trade-off. That is, we use the majority of the data for training, but we repeat the selection of the validation data several times to make sure that the validation was not just a result of outliers. The repeated division of the data into a training set and validation set can be done in different ways. For example, in *random subsampling* we just use random subsample for each set and repeat the procedure with other random samples. More common is *k-fold cross-validation*. In this technique we divide the data set into k-subsamples and use $k-1$ subsamples for training and one subsample for validation. In the next round we use another subsample for validating the training. A common choice for the number of subsamples is $k = 10$. By combining the results for the different runs we can often reduce the variance of our prediction while utilizing most data for learning.

We can sometimes help the learning process further. In many learning examples it turns out that some data are easy to learn while others are much harder. In some techniques called *boosting*, data which are hard to learn are over-sampled in the learning set so that the learning machine has more opportunities to learn these examples. A popular implementation of such an algorithm is *AdaBoost* (adaptive Boosting).

Before proceeding to general non-linear learning machines, I would like to outline a point that was recently made very eloquently by Doug Tweet in a course module that we shared last summer in a computational neuroscience course in Kingston, Canada. As discussed above, supervised learning is best phrased in terms of regression and that many applications are nonlinear in nature. It is common to make a nonlinear hypothesis in form of $y = h(\theta^T \mathbf{x})$, where θ is a parameter vector and h is a nonlinear hypothesis function. A common example of such a model is an artificial perceptrons with a sigmoidal transfer function such as $h(x) = \tanh(\theta x)$. However, as nicely stressed by Doug, there is no reason to make the functions nonlinear in the parameters which then result in a non-linear optimization problem. Support vector machines that are reviewed next are a good example where the optimization problem is only quadratic in the parameters. The corresponding convex optimization has no local minima that plagued multilayer perceptrons. The different strategies might be summarized with the following optimization functions:

$$\text{Linear Perceptron } E \propto \left(y - \theta^T \mathbf{x}\right)^2 \tag{48}$$

$$\text{Nonlinear Perceptron } E \propto (y - h(\mathbf{x};\theta))^2 \tag{49}$$

$$\text{Linear in Parameter (LIP) } E \propto \left(y - \theta^T \phi(\mathbf{x})\right)^2 \tag{50}$$

$$\text{Linear SVM } E \propto \alpha_i \alpha_j y_i y_j \mathbf{x}^T \mathbf{x} + \text{constraints} \tag{51}$$

$$\text{nonlinear SVM } E \propto \alpha_i \alpha_j y_i y_j \phi(\mathbf{x})^T \phi(\mathbf{x}) + \text{constraints} \tag{52}$$

The LIP (linear in parameters) model is more general than a linear model in that it considers functions of the form $y = \theta^T \phi(\mathbf{x})$ with some mapping function $\phi(\mathbf{x})$. In light of this review, the transformation $\phi(\mathbf{x})$ can be seen as re-coding a sensory signal into a more appropriate form with unsupervised learning methods as discussed above.

3.5 General Learning Machines

Before we leave this discussion of basic supervised learning, I would like to mention some methods which are very popular and often used for machine learning applications. In the previous section we discussed the formulation of specific hypothesis functions. However, finding an appropriate hypothesis function requires considerable domain knowledge. This is why universal learning machines have been popular with computer scientists and have a long history. A good example are artificial neural networks, specifically multilayer perceptrons, which became popular in the 1980s although they have been introduced much earlier. The general idea behind these general learning machines is to provide a very general functions with many parameters that will be adjusted through learning. Of course, the real problem is then not to over-fit the model by using appropriate restrictions and also to make the learning efficient so that it can be used to large problem size. There has been much progress in this area, specifically though the introduction of Support Vector Machines (SVMs)that I will briefly describe in this section.

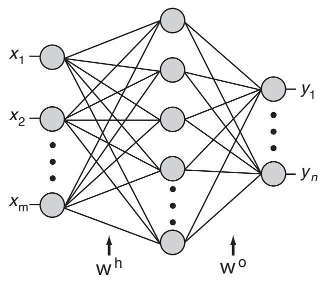

Fig. 15 Multilayer perceptron with one hidden layer. The parameters are called weights **w**.

Let us start with a multilayer perceptron as shown in Fig.15. Each node represents a simple calculation. The input layer simply relaying the inputs, while the hidden and output layer multiplay each input channel with an associated weight w_i, sum this net input, and transfer it through a generally non-linear transfer function, often chosen as a sigmoid function such as the logistic function. Such networks are thus a graphical representation of a nested nonlinear functions with parameters **w**. Applying an input results in a specific output **y** that can be compared to a desired output $\mathbf{y}_{des}$ in supervised learning. The parameters can then be adjusted, as usual in LMS regression, by minimizing the least square error $E = (\mathbf{y} - \mathbf{y}_{des})^2$, typically with gradient descent $w \leftarrow w + \alpha \frac{\partial E}{\partial w_i}$, where α is a lerning rate. Since **y** is a nested function of the parameters, this requires the applivation of the chain rule. The resulting equations look like propagating back an error term $\mathbf{y} - \mathbf{y}_{des}$ from the output to earlier layers, and this algorithm has thus been termed error-backpropagation [15].

It is easy to see that such networks are universal approximators [16], that is, the error of the training examples can be made as small as desired by increasing the number of parameters. This can be achieved by adding hidden nodes. However, the aim of supervised learning is to make predictions, that is to minimize the generalization error and not the training error. Thus, choosing a smaller number of hidden nodes might be more appropriate for this. The bias-variance tradeoff reappears here in this specific graphical model, and years of research have been investigated in solving this puzzle. There have been some good practical methods and research directions such as early stopping [17], weight decay [18] or Bayesian regularization [19] to counter overfitting, and transfer learning [20, 21] can be seen as biasing models beyond the current data set.

Most prominent are currently support vector machines (SVMs) that start by minimizing the estimated generalization (called the empirical error in this community). The main idea behind support vector machines (SVM) for binary classification is that the best linear classifier for a separable binary classification problem is the one that maximizes the margin [22, 24]. That is, there are many lines that separate the data as shown in Fig.16. The one that can be expected most robust is the one that tries to be as far from any data as possible since we can expect new data to be more likely close to the clusters of the training data if the training data are representative of the general distribution. Also, the separating line (hyperplane in higher dimensions) is determined only by a few close points that are called *support vectors*. And Vapnik's important contributions did not stop there. He also formulated the margin maximization problem in a form so that the formulas are quadratic in the parameters and only contain dot products of training vectors, $\mathbf{x}^T\mathbf{x}$ by solving the dual problem in a Lagrange formalism [22]. This has several important benefits. The problem becomes a convex optimization problem that avoids local minima which have crippled MLPs. Furthermore, since only dot products between example vectors appear in these formulations, it is possible to apply of Kernel trick to efficiently generalize these approaches to non-linear functions.

Let me illustrate the idea behind using Kernel functions for dot products. To do this it is important to distinguish attributes from features as follows. Attributes are the raw measurements, whereas features can be made up by combining attributes. For example, the attributes x_1 and x_2 could be combines in a feature vector $(x_1, x_2, x_1x_2, x_1^2, x_2^2)^T$. This is a bit like trying to guess a better representation of the problem which should be useful as discussed above with structural learning. So let us now write this transformation as function $\phi(\mathbf{x})$. The interesting part of Vapnik's formulation is that we actually do not even have to calculate this transformation explicitly but can replace the corresponding dot products as a *Kernel function*

$$K(\mathbf{x},\mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z}). \tag{53}$$

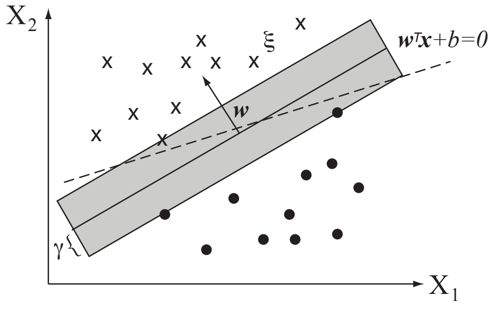

Fig. 16 Illustration of linear support vector classification.

Such Kernel functions are sometimes much easier to calculate. For example, a Gaussian Kernel function corresponds formally to an infinite dimensional feature transformation ϕ. There are some arguments from structural learning [22, 23] why SVMs are less prone to overfitting, and extensions have also been made to problems with overlapping data in form of soft margin classification [24] and to more general regression problems [25]. We will not dwell more into the theory of Suport Vector Machine but show instead an example using the popular LIBSVM [26] implementation. This implementation includes interfaces to many programming languages, such as MATLAB and Python. SVM are probably currently the most successful general learning machines.

Table 2 Using libsvm for classification

```
clear; close all; figure; hold on; axis square

%% training data and training SVM
r1=2+rand(300,1); a1=2*pi*rand(300,1); polar(a1,r1,'bo');
r2=randn(300,1); a2=.5*pi*rand(300,1); polar(a2,r2,'rx');

x=[r1.*cos(a1),r1.*sin(a1);r2.*cos(a2),r2.*sin(a2)];
y=[zeros(300,1);ones(300,1)];
model=svmtrain(y,x);

%% test data and SVM predicition
r1=2+rand(300,1); a1=2*pi*rand(300,1);
r2=randn(300,1); a2=.5*pi*rand(300,1);
x=[r1.*cos(a1),r1.*sin(a1);r2.*cos(a2),r2.*sin(a2)];
yp=svmpredict(y,x,model);

figure; hold on; axis square
[tmp,I]=sort(yp);
plot(x(1:600-sum(yp),1),x(1:600-sum(yp),2),'bo');
plot(x(600-sum(yp)+1:600,1),x(600-sum(yp)+1:600,2),'rx');
```

An example of using the LIBSVM library on data shown in Fig.17 is given in Table 2. The left figure shows training data. These data are produced from sampling two distributions. The data of the first class, shown as circles, are chosen within a ring of radius 2 to 3, while the second class, shown as crosses, are Gaussian distributed in two quadrants. These data are given with their corresponding lables to the training function `svmtrain`. The data on the right are test data. The corresponding class labels are given to the function `svmpredict` only to calculate cross validation error. For true predictions, this vector can be set to arbitrary values. The performance of this classification is around 97% with the standard parameters of the LIBSVM package. However, it is advisable to tune these parameters, for example with some search methods **??**.

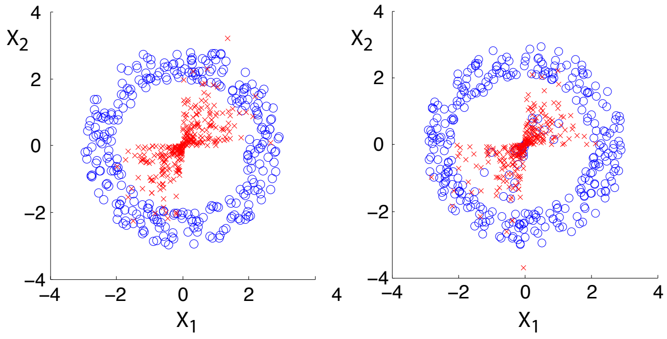

Fig. 17 Example of using training data on the left to predict the labels of the test data on the right.

4 Reinforcement Learning

As discussed above, a basic form of supervised learning is function approximation, relating input vectors to output vectors, or, more generally, finding density functions $p(\mathbf{y},\mathbf{x})$ from examples $(\mathbf{x}^{(i)},\mathbf{y}^{(i)})$. However, in many applications we do not have this kind of teacher that tells us exactly at any time the appropriate response for a specific input. Rather, feedback from a teacher is often delayed in time and also often given just in the form of general feedback such as 'good' or 'bad'. Furthermore, we are often interested in predicting a sequence of appropriate actions or an expectation of future situations from the history of past events. Thus, in this section we will talk about a more general form of temporal supervised learning, learning the temporal density function

$$p(\mathbf{y}(t+1)|\mathbf{x}^{(t)},\mathbf{x}^{(t-1)},\ldots\mathbf{x}^{(1)}). \tag{54}$$

We encountered such models already in the form of specific temporal Bayesian networks such as a Hidden Markov Process. We will now discuss this issue further within the realm of *reinforcement learning* or *learning from reward*. While we consider here mainly the prediction of a scalar utility function, most of this discussion can be applied directly to *generalized reinforcement learning*.

4.1 Markov Decision Process

Reinforcement learning can be best illustrated in a Markovian world. As discussed before, such a world[1] is characterized by transition probabilities between states, $T(s'|s,a)$, that only depend on the current state $s \in S$ and the action $a \in A$ taken from this state. We now consider feedback from the environment in the form of reward $r(s)$ and ask what actions should be taken in each state to maximize future reward. More formally, we define the *value function* or *utility function*,

$$Q^{\pi}(s,a) = E\{r(s) + \gamma r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) + ...\}_{\pi}, \tag{55}$$

as the expected future payoff (reward) when being in state s and then at s_1, s_2, etc. We introduced here the discount factor $0 \leq \gamma < 1$ so that we value immediate reward more than later reward. This is a common treatment to keep the expected value finite. An alternative scheme we would be to consider only finite action sequences. The policy $\pi(a|s)$ describes what action to take in each state. In accordance with our general probabilistic world view, we consider here probabilistic policies, that is, we want to know with which probability an action should be chosen. If the policies are deterministic, then taking a specific action is determined by the policy, and the value function is then often written as $V^{\pi}(s)$.[2] Our goal is to find the optimal policy, which is the policy that maximizes the expected future payoff,

$$\pi^*(a|s) = \arg\max_{\pi} Q^{\pi}(s,a), \tag{56}$$

where the argmax function picks the policy for which Q is maximal. Such a setting is called a *Markov Decision Process (MDP)*.

MDPs have been studied since the mid 1950s, and Richard Bellman noted that it is possible to estimate the value function for each policy π from a self-consistent equation now called *Bellman equation*. He called the corresponding algorithm *dynamic programming*. Specifically, we can separate the expected value of the immediate reward from the expected value of the reward from visiting subsequent states,

$$Q^{\pi}(s,a) = E\{r(s)\}_{\pi} + \gamma E\{r(s_1) + \gamma r(s_2) + \gamma^2 r(s_3) + ...\}_{\pi}. \tag{57}$$

[1] Most often we talk about Markov models where the model is a simplification or abstraction of a real world. However, here we discuss a specific toy world in which state transitions fulfill the Markov condition.

[2] $V^{\pi}(s)$ is usually called the state value function and $Q^{\pi}(s,a)$ the state-action value function. However, note that the value depends in both cases on the states and the actions taken.

The second expected value on the right hand side is that of the value function for state s_1. However, state s_1 is related to state s since state s_1 is the state that can be reached with a certain probability from s when taking action a_1 according to policy π (e.g., $s_1 = s + a_1$, or more generally $s_n = s_{n-1} + a_n$). We can incorporate this into the equation by writing

$$Q^\pi(s,a) = r(s) + \gamma \sum_{s'} T(s'|s,a) \sum_{a'} \pi(a'|s') E\{r(s') + \gamma R(s'_1) + \gamma^2 R(s'_2) + ...\}_\pi, \quad (58)$$

where s'_1 is the next state after state s', etc. Thus, the expression on the right is the state-value-function of state s'. If we substitute the corresponding expression of equation (55) into the above formula, we get the *Bellman equation* for a specific policy, namely

$$Q^\pi(s,a) = r(s) + \gamma \sum_{s'} T(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s',a'). \quad (59)$$

The action a is uniquely specified by the policy in the case of deterministic policies, and the value function $Q^\pi(s,a)$ reduces to $V^\pi(s)$. In this case the equation simplifies to

$$V^\pi(s) = r(s) + \gamma \sum_{s'} T(s'|s,a) V^\pi(s'). \quad (60)$$

This formulation of the Bellman equation for the MDP [28, 29, 30] is slightly different to the formulation of Sutton and Barto in [31] since the later used a slightly different definition of the value function as the cumulative reward from the next state only and not the current state[3]. The corresponding Bellman equation is then

$$V^\pi(s) = \sum_{s'} T(s'|s,a)(r(s') + \gamma V^\pi(s')). \quad (61)$$

The Bellman equation is a set of N linear equations in an environment with N states, one equation for each unknown value function of each state. Given that the environment is known, that is, knowing functions r and T, we can use well known methods from linear algebra to solve for $V^\pi(s)$. This can be formulated compactly with Matrix notation,

$$\mathbf{r} = (\mathbb{1} - \gamma \mathbf{T})\mathbf{V}^\pi, \quad (62)$$

where $\mathbf{r}$ is the reward vector, $\mathbb{1}$ is the unit diagonal matrix, and $\mathbf{T}$ is the transition matrix. To solve this equation we have to invert a matrix and multiply this with the reward values,

$$\mathbf{V}^\pi = (\mathbb{1} - \gamma \mathbf{T})^{-1} \mathbf{r}^t, \quad (63)$$

where $\mathbf{r}^t$ is the transpose of $\mathbf{r}$. It is also common to use the Bellman equation directly and calculate a state-value-function iteratively for each policy. We can start with a guess $\mathbf{V}$ for the value of each state, and calculating from this a better estimate

[3] This is just a matter of when we consider the prediction, just before getting the current reward of after taking the next step.

$$\mathbf{V} \leftarrow \mathbf{r} + \gamma \mathbf{T}\mathbf{V} \tag{64}$$

until this process converges. This then gives us a value function for a specific policy. To find the best policy, the one that maximizes the expected payoff, we have to loop through different policies to find the maximal value function. This can be done in different ways. Most commonly it is to use *Policy iteration*, which starts with a guess policy, iterates a few times the value function for this policy, and then chooses a new policy that maximizes this approximate value function. This process is repeated until convergence.

It is also possible to derive a version of *Bellman's equation for the optimal value function* itself,

$$V^*(s) = r(s) + \max_a \gamma \sum_{s'} T(s'|s,a) V^*(s'). \tag{65}$$

The max function is a bit more difficult to implement in the analytic solution, but we can again easily use an iterative method to solve for this optimal value function. This algorithm is called *Value Iteration*. The *optimal policy* can always be calculated from the optimal value function,

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s'|s,a) V^*(s'). \tag{66}$$

A policy tells an agents what action should be chosen, and the optimal policy is hence related to optimal control as long as the reward reflects the desired performance.

The previously discussed policy iteration has some advantages over value iterations. In value iteration we have to try out all possible actions when evaluating the value function, and this can be time consuming when there are many possible actions. In policy iteration, we choose a specific policy, although we have then to iterate over consecutive policies. In practice it turns out that policy iteration often converges fairly rapidly so that it becomes a practical method. However, value iteration is a little bit easier and has more similarities to the algorithms discussed below that are also applicable to situations where we do not know the environment a priori.

4.2 Temporal Difference learning

In dynamic programming, the agent goes repeatedly to every possible state in the system. This can be time consuming. It usually works if we have complete knowledge of the system since we do not really perform the actions but can sit and calculate the solution in a planning phase. However, we might not know the rewards given in different states, or the transition probabilities, etc. One approach would be to estimate these quantities from interacting with the environment before using dynamic programming. The following methods are more estimations of the state value function that determines optimal actions. These online methods assume that we still know exactly in which state the agent is. We will consider observable situations later.

A general strategy for estimating the value of states, called *Monte Carlo methods*, is to act in the environment and thereby to sample and memorize reward from which the expected value can be calculated by simple averaging. Such ideas can be combined with the bootstrapping ideas of dynamic programming. The resulting algorithms are generally called *temporal difference (TD) learning* since they rely on the difference between expected reward and actual reward.

We start again by estimating the value function for a specific policy before moving to schemas for estimating the optimal policy. Bellman's equations require the estimation of future reward

$$\sum_{s'} T(s'|s,a)V^{\pi}(s') \approx V^{\pi}(s'). \tag{67}$$

In this equation we introduced an approximation of this sum by the value of the state that is reached by one Monte Carlo step. In other words, we replace the total sum that we could build when knowing the environment with a sampling step. While this approach is only an estimation, the idea is that this will still result in an improvement of the estimation of the value function, and that other trials have the possibility to evaluate other states that have not been reached in this trial. The value function should then be updated carefully, by considering the new estimate only incrementally,

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\{r(s) + \gamma V^{\pi}(s') - V^{\pi}(s)\}. \tag{68}$$

This is called *temporal difference* or *TD learning*. The constant α is a learning rate and should be fairly small. This policy evaluation can then be combined with policy iteration as discussed already in the section on dynamic programming.

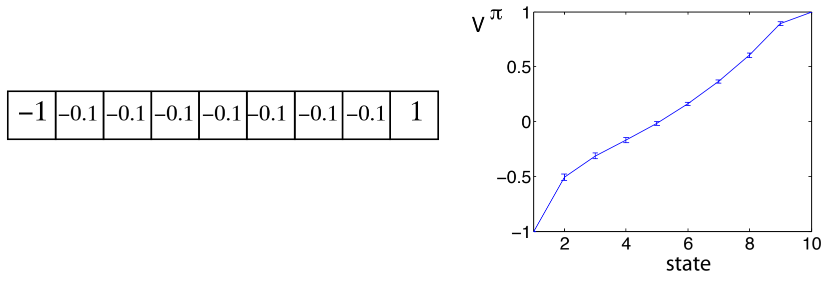

Fig. 18 Example of using TD learning on a chain of rewarded states. The right graph shows the estimated value function over 100 runs. The errorbars depict the standard deviations.

An example program is given in Table 3. The state space in this example consists of a chain of 10 states as shown on the left in Fig.18. The 10th state is rewarded with

Table 3 TD learning with a chain state space

```
% Chain example: Policy iteration with TD learning
clear; N=10; P=0.8; gamma=0.9; % parameters
r=zeros(1,N)-0.1; r(1)=-1; r(N)=1; % reward function

% transition probabilities; 1 (going left) and 2 (going right)
T=zeros(N,N,2); %T(1,1,:)=1; T(N,N,:)=1; %Absorbing end states
for i=2:N-1;
    T(i,i-1,1)=P; T(i,i+1,1)=1-P; T(i,i-1,2)=1-P; T(i,i+1,2)=P;
end

% initially random start policy and value function
policy=floor(2*rand(1,N))+1; Vpi=rand(N,1);

for iter=1:10 %policy iteration
    Vpi(1)=0; Vpi(N)=0; %absorbing end states
    %Transition matrix of choosen action
    for s=2:N-1;
        Tpi(s,s-1)=T(s,s-1,policy(s));
        Tpi(s,s+1)=T(s,s+1,policy(s));
    end
    % Estimate V for this policy using TD learning
    for i=1:100 %loop over episodes
        s=floor(9*rand)+1; %initial random position
        while s>1 && s<N
            a=policy(s); %choose action
            s1=s+2*(a-1)-1; %calculate next state
            if rand>Tpi(s,s1); s1=s-2*(a-1)+1; end %random execution
            Vpi(s)=Vpi(s)+0.01*(r(s1)+gamma*Vpi(s1)-Vpi(s)); %update
            s=s1;
        end
    end
    Vpi(1)=r(1); Vpi(N)=r(N);
    % Updating policy
    policy(1)=0; policy(N)=0; %absorbing states
    for s=2:N-1
        [tmp,policy(s)] = max([Vpi(s-1),Vpi(s+1)]);
    end
end
```

$r = 1$, while the first state receives a large negative reward, $r = -1$. The intermediate states receive a small negative reward to account for movement costs. The estimated value function after 10 policy iteration is shown on the right. This curve represents the mean of 100 such simulations, and the standard deviation is shown as errorbar.

One of the greatest challenges in this approach is the potential conflict between taking a step that provides a fairly large expected reward and exploring an unknown territory to potentially find larger rewards. This *exploration-exploitation dilemma* will now be addressed with stochastic policies and will thus return to the notation of the state-action value function.[4] To include exploration in the algorithm, we include some randomness in the policy. For example, we could follow most of the time the *greedy policy* and only choose another possible action in a small number ε of times. This probabilistic policy is called the *ε-greedy policy*,

$$\pi(a = \arg\max_a Q(s,a)) = 1 - \varepsilon. \tag{69}$$

This policy is choosing the policy with the highest expected payoff most of the time while treating all other actions the same. A more graded approach employs the *softmax policy* which choses each action proportional to a Boltzmann distribution

$$\pi(a|s) = \frac{e^{\frac{1}{T}Q(s,a)}}{\sum_{a'} e^{\frac{1}{T}Q(s,a')}}. \tag{70}$$

This policy choses most often the one with highest expected reward, followed by the second highest, etc, and the temperature parameter T sets the relative probability of these choices. Temporal difference learning for the optimal value function with stochastic policies is

$$Q(s,a) \leftarrow Q(s,a) + \alpha\{r(s) + \gamma Q(s',a') - Q(s,a)\}, \tag{71}$$

where the actions a' is the action chosen according to the policy. This *on-policy TD algorithm* is called *Sarsa* for state-action-reward-state-action [31]. A slightly different approach is using only the action to the maximal valued state for the value function update while still exploring the state space through the stochastic policy,

$$Q(s,a) \leftarrow Q(s,a) + \alpha\{r(s) + \max_{a'} \gamma Q(s',a') - Q(s,a)\}. \tag{72}$$

Such an *off-policy TD algorithm* is called **Q**-*leaning*. These algorithms have been instrumental in the success of reinforcement learning in many engineering applications.

4.3 Function approximation and TD(λ)

There are several challenges in reinforcement learning remaining. Specifically, the large number of states in real world applications makes these algorithms unpractical. This was already noted by Richard Bellman who coined the phrase *curse of dimensionally*. Indeed, we have only considered so far discrete state spaces, while many applications are in a continuous state space. While discretizing a continuous state space is a common approach, increasing the resolution of the discretization will increase the number of states exponentially. Another major problem in practice

[4] We will drop the star in the notation for the optimal value function since we will now only consider optimal value functions.

is that the environment is not fully or reliably observable. Thus, we might not even know exactly in which state the agent is when considering the value update. A common approach to the partially observable Markov decision process (POMDP) is the introduction of a probability map. In the update of the Bellman equation we need then to consider all possible states that can be reached from the current state, which will typically increase the number of calculations even further. We will thus not follow this approach here and consider instead the use of function approximators to overcome these problems.

The idea behind the following method is to make a hypothesis of the relation between sensor data and expected values in form of a parameterized function as in supervised learning,[5]

$$V_t = V(\mathbf{x}_t) \approx V(\mathbf{x}_t; \theta), \tag{73}$$

and estimate the parameters by maximum likelihood as before. We used thereby a time index to distinguish the sequences of states. In principle, one could build very specific temporal Bayesian models for specific problems as discussed above, but I will outline here again the use of general learning machines in this circumstance. In particular, let us consider adjusting the weights of a neural network using gradient-descent methods on a mean square error (MSE) function

$$\Delta\theta = \alpha \sum_{t=1}^{m} (r - V_t) \frac{\partial V_t}{\partial \theta}. \tag{74}$$

We considered here the total change of the weights for a whole episode of m time steps by summing the errors for each time step. One specific difference of this situation to the supervised learning examples before is that the reward is only received after several time steps in the future at the end of an episode. One possible approach for this situation is to keep a history of our predictions and make the changes for the whole episode only after the reward is received at the end of the episode. Another approach is to make incremental (online) updates by following the approach of temporal difference learning and replacing the supervision signal for a particular time step by the prediction of the value of the next time step. Specifically, we can write the difference between the reward and the prediction at the end of the sequence at time t when reward is received, $V_{t+1} = r$, as

$$r - V_t = \sum_{k=t}^{m} (V_{k+1} - V_k) \tag{75}$$

since the intermediate terms chancel out. Using this in equation (74) gives

$$\Delta\theta = \alpha \sum_{t}^{m} \sum_{k=t}^{m} (V_{k+1} - V_k) \frac{\partial V_t}{\partial \theta} \tag{76}$$

[5] The same function name is used on both sides of the equation but these are distinguished by inclusion of parameters. The value functions below refer all to the parametric model, which should be clear from the context.

$$= \alpha \sum_{t=1}^{m} (V_{t+1} - V_t) \sum_{k=1}^{t} \frac{\partial V_k}{\partial \theta}, \tag{77}$$

which can be verified by writing out the sums and reordering the terms. Of course, this is just rewriting the original equation 74. We still have to keep a memory of all the gradients from the previous time steps, or at least a running sum of these gradients.

While the rules (74) and (77) are equivalent, we also introduce here some modified rules suggested by Richard Sutton [32]. In particular, we can weight recent gradients more heavily than gradients in the more remote past by introducing a decay factor $0 \leq \lambda \leq 1$. The rule above correspond to $\lambda = 1$ and is thus called the *TD(1) rule*. The more general *TD(λ) rule* is given by

$$\Delta_t \theta = \alpha (V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial V_k}{\partial \theta}. \tag{78}$$

It is interesting to look at the extreme of $\lambda = 0$. The *TD(0) rule* is given by

$$\Delta_t \theta = \alpha (V_{t+1} - V_t) \frac{\partial V_t}{\partial \theta}. \tag{79}$$

While this rule gives gives in principle different results with respect to the original supervised learning problem described by TD(1), it has the advantage that it is local in time, does not require any memory, and often still works very well. TD(λ) algorithm can be implemented with a multilayer perceptron when back-propagating the error term to hidden layers. A generalization to stochastic networks has been made in the free-energy formalism [33].

4.4 An example of TD learning

5 Some biological analogies

The brain seems to be a very successful learning machine, and it is therefore not surprising that human capabilities have motivated much research in artificial intelligence. But also, insights from learning theory are important for our understanding of brain processes. In this final section I want to mention some interesting relations of neuroscientific issues with learning theory. I already remarked on the close relation between unsupervised learning and receptive fields in the early sensory areas of the cortex that I believe is a wonderful example of underlying mechanisms behind physiological findings. In the following I would like to add remarks on two further subjects related to supervised learning and to reinforcement learning. The first is about synaptic plasticity which seems to be an important mechanisms for the physical implementation of learning rules. The second is about the close relation of reinforcement learning with classical conditioning and the Basal Ganglia. Classical conditioning has been a major area in animal learning, and recent recordings in the Basal Ganglia has helped relating these areas on a behavioural, physiological and learning theoretical level.

5.1 Synaptic Plasticity

As speculated by the Canadian Donald Hebb [34], the leading theory of the physical implementation of learning is that of synaptic changes where the synaptic efficacy changes in response to causally related pre- and post synaptic firings. Such correlation rules have first been made concrete by Eduardo Caianello [35] ad have recently been made more specific in terms of spike-timing-dependent-platicity (STDP; see for example [36]). The principle idea is that when a driving neuron participates in firing a subsequent neuron, then the connection strength between these neurons will increase and decrease otherwise. All the learning rules in this chapter have followed this main association rule with terms that are proportional to pre- and post-synaptic firing. Synaptic plasticity is not only a fascinating area in neuroscience but has also important medical implications since neurodegenerative deceases such as Alzheimer's decease and dementia have synaptic implications and several drugs act on synaptic receptors.

There are many mysteries left that need to be understood if we want to make progress in helping with neurological conditions. One basic fact that seems puzzling is that synapses are not long lasting compared to the time scale of human memories[6]. Synapses consist of proteins that have to be actively maintained by protein synthesis. Thus, it is puzzling how this maintenance can survive years and aiding memories such as when returning to a place of birth after many years of absence or meeting friends that we have not seen for years? These are very basic questions that have, to my knowledge, not been addressed sufficiently.

There are other aspects in synaptic plasticity that seems well represented in many models. In particular, I would like to point the findings of my friend Alan Fine and his colleagues that fits nicely with the probabilistic theme that I have tried to stress in this chapter. Fine and colleagues have performed classical LTP/LTD experiments that use high or low frequency stimulations of hippocampal slices of rodents to induce LTP and LTD, respectably. Some of their results are summarized in Fig.19. To test the strength of the synapses, they stimulated with two pulses as paired pulses facilitate synaptic responses (the second pulse makes it easier to elicit a postsynaptic spike). The slices are then activated with high frequency stimulations in-between these test. As shown in Fig.19A, the electric response of the postsynaptic neuron as measure by the excitatory post-synaptic potential (EPSP) is higher after the high frequency stimulation. This corresponds to the classical findings by Bliss and Lomo [37]. Fine and colleagues also imaged the calcium-related optical luminance signal

[6] Julian Miller made this point nicely at the workshop

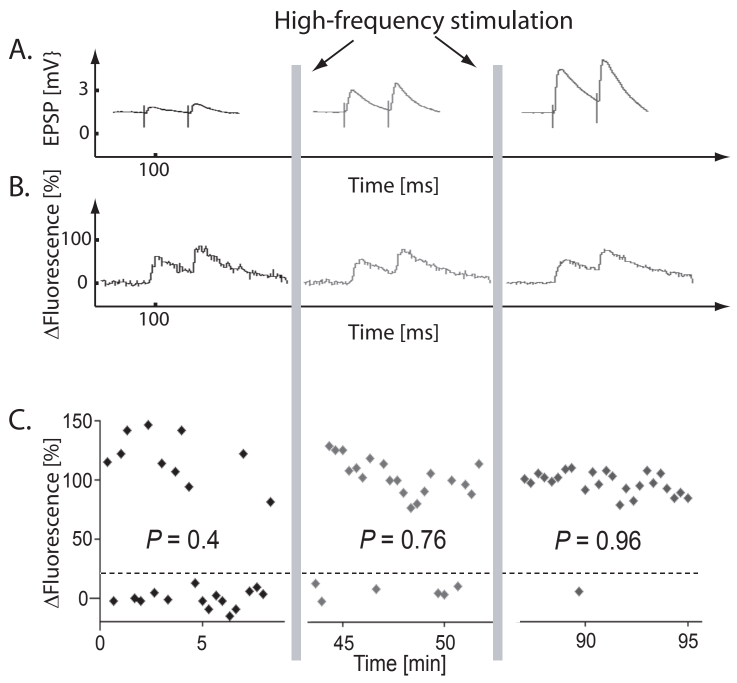

Fig. 19 Plasticity experiment in hippocampal slices in which not only EPSPs were measured, but in which, additionally, postsynaptic calcium-dependent fluorescence signals at single synapses were imaged. [Data courtesy of Alan Fine and Ryosuke Enoki].

from individual synapses, shown in Fig.19B, and found that this luminance did not change despite the fact the calcium-dependent mechanisms are generally associated with plasticity. Instead, they found that the probability of eliciting a postsynaptic spike varied nicely along the lines of experimental conditions classically related to LTP/LTD since a low frequency stimulus also lead to a decrease in EPSP and a decrease in the probability of synaptic response (not shown in the figure).

A manipulation of the probability of transmitter release could explain the increased EPSP in such experiments. If there is a population of synapses that drive that neuron, than a population of synapses with higher likelihood of transmitter release would result in a larger EPSP than a population with smaller likelihood of transmitter release. In this sense, the findings are still consistent with some of the consequences of synaptic plasticity. But these findings also point to additional possibilities which are consistent with the view that brain processing might be based on probabilistic calculus rather than dealing with point estimates. Thus, the common view of a noisy nervous system with noisy synapses might be misleading. If this is noise in the sense of limitation of biological implementations, why could the probability of synaptic responses be modulated reliably?

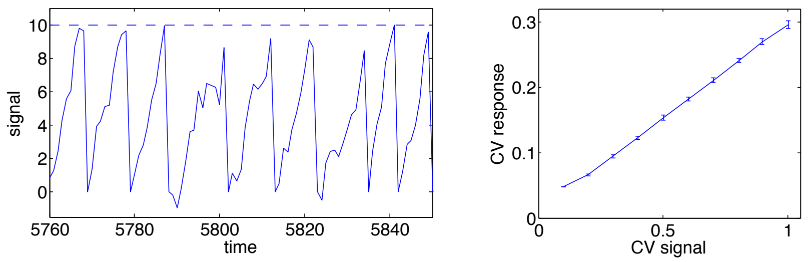

Fig. 20 Demonstration of the relation between variability in signal versus variability in spike timing response.

From a theoretical perspective it is rather difficult that noise survives thresholding processes. For example, consider a biased random walk to a threshold with unbiased Gaussian noise as illustrated on the left in Fig.20. The noise in the process leads to different times of threshold crossings and the variation of the threshold crossing time is related to the variations in the signal as shown on the right in Fig.20 where $C_v = \sigma/\mu$ is the coefficient of variation. While there is a positive slope between them (higher noise leads to higher variations in firing times), the proportionality factor is only around $1/\sqrt{4\pi}$. Thus, if noise is an issue, then one could use thresholding mechanisms to reduce it, and, with repeated stages as in the brain the noise, should become smaller. In other words, if noise is the problem then filter it out early and higher processes should be less noisy. Thus, it is likely that the variations in the brain are not all noise but could have important information processing roles such as representing the likelihood of sensory signals or the confidence in possible actions. This is consistence with the probabilistic approaches to machine learning.

5.2 *Classical Conditioning and the Basal Ganglia*

One of the important roles of computational neuroscience is bridging the gap between behavioural and physiological findings [38]. The following discussion is a good example. Classical conditioning has been studied intensely in the psychological discipline of animal learning at least since the studies by Pavlov. One of the most basic findings of Pavlov is that it is possible to learn that a stimulus is predicting reward and that this prediction elicits the same behaviour as the primary reward signal like salivation following a tone when the tone predicts food reward. Many similar predictions have been summarized very successfully by the Rescorla-Wagner theory [40]. In terms of learning theories as discussed above, this theory relates the change in the value of a state ΔV_i to the reward prediction error $\lambda - V_i$,

$$\Delta V_i = \alpha_i \beta (\lambda - V_i), \tag{80}$$

where α_i and β describe the saliencies of the conditioned stimulus and the unconditioned stimulus, receptively, and λ corresponds to reward. This model correspond to temporal difference learning in a one step prediction task where the reward follows immediately the stimulus.

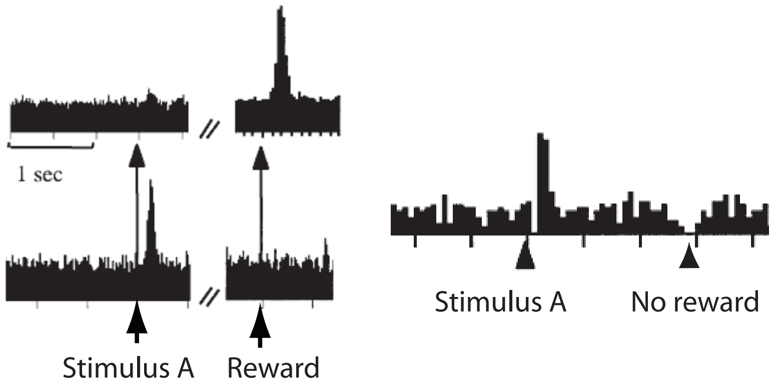

Fig. 21 Recordings by Schultz et al. [39] in a classical conditioning experiment. I these experiments a stimulus was presented followed by a reward. Early in the trials the SN neurons responded after the animal received a reward (top left), while the neurons responded to the predictor of the reward in later trials (bottom left). The neurons even seem to indicate a absence of an expected reward after learning (right).

The Rescola-Wagner theory with its essential reliance on the reward prediction error is very successful in explaining behaviour, and was very exciting when Wolfram Schultz [39] and colleagues discovered neural signatures of reward prediction errors. Schultz found these signals in the Substantia Nigra, which is part of a complex of different nuclei in the midbrain called the Basal Ganglia. The name 'Substantia Nigra' means 'black substance', and the blackness of this area is apparently due to chemical compound related to dopamine since these neurons transmit dopamine to the input area of the Basal Ganglia and to the cortex, and Dopamine has been implicated with modulating learning. Some examples of the response of these neurons are shown in Fig.21.

We can integrate the Rescorla-Wagner theory with these physiological findings in a neural network model as shown in Fig.22. The reward prediction error, $\hat{r}$ is conveyed by the nigra neurons to the striatum to mediate the plasticity of cortical-striatal synapses. The synapses are thereby assumed to contain an eligibility trace since the learning rule requires the association with the previous state. Many psychological experiments can be modelled by a one-step prediction task where the actual reward follows a specific condition. The learning rule then simplifies to a temporal learning rule where the term with γ can be neglected, which corresponds to the model in Fig.22A. The implementation of the full TD rule would then require a fast side loop as shown in Fig.22B, which has been speculated to be associated with the subthalamus [42].

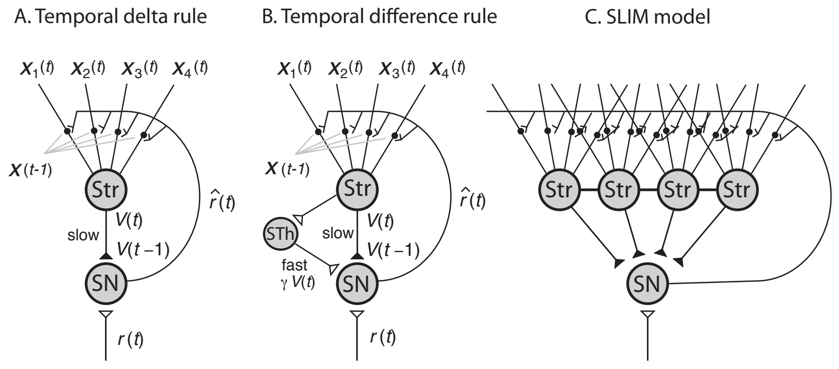

Fig. 22 Implementation of reinforcement learning models with analogies to the Basal Ganglia. (A) Single state of one-step reinforcement learning model (temporal delta rule) with cortical input, a Striatal neuron (Str), and a neuron in the substantia nigra (SN) that conveys the reward prediction error to striatal spines. (B) Implementation of the temporal difference (TD) learning with a fast subthalamic side loop. (C) Basic version of the Striatal-with-lateral-inhibtion (SLIM) model.

Of course, the anatomy of the Basal ganglia is more elaborate. My student Patrick Connor and I have suggested a model with lateral interactions in the striatum [43], that has some physiological grounding [41] and can explain a variety of behavioral that are not covered by the Rescorla Wagner model [44]. Also, there are two main pathways through the Basal Ganglia, a direct pathway and an indirect one, with intermediate stages in distinct subregions of the Basal Ganglia (not shown in Fig.22). The direct pathway has a facilitory effect on the output neurons of the Basal Ganglia, and the indirect has an inhibitory effect. Since the effect of the output of the Basal Ganglia is itself inhibiting motor areas, it has been speculated that the direct pathway could learn to inhibit non-rewarding actions, whereas the indirect pathway could learn to support learning of facilitating rewarding actions. Different alterations of specific pathways has been suggested to relate to different neurological conditions that are known to have relations to the Basal Ganglia such as Parkinson decease, Tourette syndrome, ADHD, schizophrenia and others [45]. Thus, modelling and understanding this learning system has the potential to guide refined intervention strategies.

6 Outlook

Learning is an exciting field that has made considerable progress in the last few years, specifically though statistical learning theory and its probabilistic embedding.

These theories have at least clarified what could be expected with ideal learning systems such as their ability to generalize. Much progress has also been made with unsupervised and to start tackling temporal learning problems. Most excitingly, the progress in this area enabled machine learning to find its way out of the research labs and into commercial products that revolutionize technologies. Statistical learning theory has clarified general learning principles such as the optimal generalizability and optimal (Bayesian) decision making in the face of uncertainties.

What then are the outstanding questions? While machine learning has enabled interesting applications, many of these applications are very focused in scope. The complexity of the environments that faces humans seems still far beyond our reach. Scaling up these methods even further is important to enable even more applications. Many believe that for this we require truly hierarchical systems [46], and specifically such systems that processes temporal data [47]. While there is exciting progress in this field, learning to map simple features, such as pixels from an image, to high level concepts, such as objects in a scene, is still challenging.

While Bayesian inference has been instrumental the maturation of machine learning, there are also many limitations of such methods. Specifically, truly Bayesian methods have an seemingly unbounded requirement for knowledge as we typically have to sum over all possible outcomes with their likelihood of such events. This seems not only excessive in it's required knowledge and processing demands, but also faces practical limitations for many applications.

An alternative approach is bounded rationality that might underly a lot hove human decision making [48]. Critical for the success of such methods are fast and frugal heuristics that depend on the environment. Thus, there is a major role for learning in this domain on many different scales, including developmental and genetic domains. Indeed, there is now an increased realization of environmental mechanisms that influence genetic decoding. This field of epigenetic is very promising in discovering new mechanisms of environmental responsive systems.

References

1. G. Hinton (2002) Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14, 1711–1800.
2. A. Graps, An Introduction to Wavelets, http://www.amara.com/IEEEwave/IEEEwavelet.html.
3. N. Huang et al. (1998) The empirical mode decomposition and the Hilbert spectrum for non-linear and non-stationary time series analysis. Proc. R. Soc. Lond. A (1998) 454, 903995.
4. P. Smolensky (1986) Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart, D. E. and McClelland, J. L., editors, Parallel Distributed Processing: Volume 1: Foundations, pages 194-281. MIT Press, Cambridge, MA.
5. G. Hinton (2010). A Practical Guide to Training Restricted Boltzmann Machines. *U. of Toronto Technical Report UTML TR 2010-003*.
6. H. Barlow (1961) Possible principles underlying the transformation of sensory messages. Sensory Communication, pp. 217-234, 1961.
7. P. Fldik (1990), Forming sparse representations by local anti-Hebbian learning, Biological Cybernetics, vol. 64, pp. 165-170.
8. P. Fldik and D. Endres (2008) Sparse coding. *Scholarpedia*, 3, 2984.

9. B. Olshausen and D. Field (1996) Emergence of Simple-Cell Receptive Field Properties by Learning a Sparse Code for Natural Images, Nature, 381: 607-609.
10. H. Lee, E. Chaitanya and A. Ng (2007) Sparse deep belief net model for visual area V2, NIPS*2007
11. T. Kohonen (1994). Self-Organizing Maps, Springer.
12. P. Hollensen, P. Hartono and T. Trappenberg (2011) Topographic RBM as Robot Controller, JNNS 2011.
13. S. Chatterjee and A. Hadi (1988) Sensitivity Analysis in Linear Regression. John Wiley & Sons: New York.
14. Judea Pearl (2009) Causality: Models, Reasoning and Inference, Cambridge University Press.
15. D. Rumelhart, G. Hinton and R. Williams (1986) Learning representations by back-propagating errors, Nature 323(6088): 533536.
16. K. Hornik (1991) Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2): 251257.
17. A. Weigend, D. Rumelhart (1991) Generalization through Minimal Networks with Application to Forecasting, In: Computing Science and Statistics (23rd Symposium INTERFACE'91, Seattle, WA), edited by E. M. Keramidas, 362–370.
18. R. Caruana, S. Lawrence and C. Lee Giles (2000) Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping, in Proc. Neural Information Processing Systems Conference, 402–408.
19. D.J.C. MacKay (1992), A Practical Bayesian Framework for Backpropagation Networks. Neural Computation, Vol. 4(3): 448-472.
20. D. Silver and K. Bennett (2008) Guest editor's introduction: special issue on inductive transfer learning. Machine Learning 73(3): 215-220.
21. S. Pan, Q. Yang (2010) A Survey on Transfer Learning, IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE), 22(10):1345-1359.
22. V. Vapnik (1995), The Nature of Statistical Learning Theory, Springer.
23. C. Burges (1998) A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery, 2(2), 121167.
24. C. Cortes and V. Vapnik (1995) Support-Vector Networks, Machine Learning 20: 273-297
25. A. Smola and B. Schölkopf (2004) A tutorial on support vector regression, Statistics and Computing 14(3).
26. C.-C. Chang and C.-J. Lin (2001) LIBSVM: a library for support vector machines, Software available at http://www.csie.ntu.edu.tw/?cjlin/libsvm
27. M. Boardman, T. Trappenberg (2006) A Heuristic for Free Parameter Optimization with Support Vector Machines, WCCI 2006, 1337-1344. Source code at http://www.cs.dal.ca/∼boardman/wcci.
28. Ethem Alpaydim (2010) Introduction to Machine Learning, 2e, MIT Press.
29. S. Thrun, W. Burgard, and D. Fox (2005) Probabilistic Robotics. MIT Press.
30. S. Russel and P. Norvig (2010) Artificial Intelligence: A Modern Approach, Third Edition, Prentice Hall.
31. Richard S. Sutton and Andrew G. Barto (1998) Reinforcement Learning: An Introduction, MIT Press.
32. R. Sutton (1988) Learning to predict by the methods of temporal differences. Machine Learning 3: 9-44, erratum p. 377.
33. B. Sallans G. Hinton (2004), *Reinforcement Learning with Factored States and Actions*, Journal of Machine Learning Research 5: 1063–1088.
34. D.O. Hebb (1949) The Organization of Behaviour, John Wiley & Sons.
35. E.R.Caianiello (1961) Outline of a theory of thought-processes and thinking machines, J. Theor. Biol. 1: 204–235
36. T. Trappenberg (2010) Fundamentals of Computational Neuroscience, 2nd edition, Oxford University Press
37. T. Bliss and T. Lomo (1973) Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path, J. Physiol. 232(2): 331–56. .

38. D. Heinke and E. Mavritsaki (eds.) (2008) Computational Modelling in Behavioural Neuroscience: Closing the gap between neurophysiology and behaviour, Psychology Press, London
39. W. Schultz (1998) Predictive Reward Signal of Dopamine Neurons, J. Neurophysiol. 80(1):1–27.
40. R. Rescorla and A. Wagner (1972) A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In P. W. Black AH (Ed.), Classical conditioning ii: Current research and theory (pp. 6499). New York: Appleton Century Crofts.
41. J. Reynolds and J. Wickens (2002) Dopamine-dependent plasticity of corticostriatal synapses. Neural Networks, 15(4-6), 507521.
42. J. Houk, J. Adams and A. Barto (1995) A model of how the Basal Ganglia generate and use neural signals that predict reinforcement, in Models of information processing in the Basal Ganglia, Hauk, Davis and Breiser (eds.), MIT Press.
43. P. Connor and T. Trappenberg (2011), Characterizing a Brain-Based Value-Function Approximator, in Advances in Artificial Intelligence LNAI 2056, Eleni Stroulia and Stan Matwin (eds), Springer 2011, 92-103.
44. P. Connor, V. LoLordo, and T. Trappenberg (2012) A Striatal Model of Pavlovian Conditioning, submitted.
45. T. Maia and M. Frank (2011) From reinforcement learning models to psychiatric and neurological disorders, Nature Neuroscience 14:154 - 162
46. Y. Bengio (2009) Learning Deep Architectures for AI. Found. Trends Mach. Learn. 2:1-127.
47. J. Hawkins (2004) On intelligence, Times Books.
48. G. Gigerenzer, P. Todd and the ABC Research Group (1999) SImple Heuristics that make us smart, Oxford University Press.