EPJ Web of Conferences **55**, 02003 (2013)
DOI: 10.1051/epjconf/20135502003

Introduction to neural networks in high energy physics

Jan Therhaag

[1] *Rheinische Friedrich-Wilhelms-Universität Bonn*

Abstract. Artificial neural networks are a well established tool in high energy physics, playing an important role in both online and offline data analysis. Nevertheless they are often perceived as *black boxes* which perform obscure operations beyond the control of the user, resulting in a skepticism against any results that may be obtained using them. The situation is not helped by common explanations which try to draw analogies between artificial neural networks and the human brain, for the brain is an even more complex black box itself. In this introductory text, I will take a problem-oriented approach to neural network techniques, showing how the fundamental concepts arise naturally from the demand to solve classification tasks which are frequently encountered in high energy physics. Particular attention is devoted to the question how probability theory can be used to control the complexity of neural networks.

1 Introduction: A simple classification problem

Two-class classification problems are among the tasks most frequently encountered in high energy physics data analysis, usually in the form of separating interesting data (*signal*) from unwanted noise (*background*). For our discussion of neural networks, we will always consider the situation where simulated data (referred to as *Monte Carlo*) for both signal and background is available in the form of separate sets of independent events with a fixed number of characteristic features (*variables*). The Monte Carlo enables us to learn distinctive features of both event species which may then be used to discriminate the signal against the background when analyzing the observed data. The data set used for feature extraction is often called *training data*. The Monte Carlo also serves as pseudo data (*test data*) on which the discriminating function we have inferred from the training data set can be assessed. Since the true class label (signal or background) is available for Monte Carlo, one can easily determine the fraction of events in the test data set that is correctly identified. To obtain an unbiased estimate of this fraction, it is of course important that the training data set and the test data set are statistically independent.

For simplicity we consider the training data set shown in the left panel of figure 1. Data points in this example are fully described by their position in the (x_1, x_2)-plane. Let's assume that the orange points are drawn from the signal distribution while the blue points are drawn from the background distribution. The black line separating the two shaded areas constitutes the optimal *decision boundary* between the two classes, which could be calculated if the underlying distributions for signal and background were known. Since this is usually not the case, we have to try to infer it from the training data. A very simple approach to distinguishing the two classes would be to examine the distributions of the variables x_1 and x_2 separately and perform what is known as a cut analysis. We can however

find a better solution by taking into account linear combinations of the two variables: let's assign a binary label to each data point and use these labels as target values to fit a linear function to the data points using a simple least-squares approach:

$$\hat{a}(\mathbf{x}) = \sum_{i=1}^{2} w_i x_i + w_0 \tag{1}$$

The parameters of this model are the coefficients w_0, w_i which are adjusted to give the best fit of the linear function to the training data. If we choose to label the signal events (orange color) with 1 and the background events with -1, we find the solution shown in the right panel of figure 1. We may now call $\{\mathbf{x}|\hat{a}(\mathbf{x}) > 0\}$ the signal-like region while $\{\mathbf{x}|\hat{a}(\mathbf{x}) < 0\}$ is called the background-like region. Note that the decision boundary defined by $\{\mathbf{x}|\hat{a}(\mathbf{x}) = 0\}$ is linear as would be expected from the model. The function $\hat{a}$ can now be used to select signal candidate events from newly observed data by accepting only those which fulfill $\hat{a}(\mathbf{x}) > 0$.

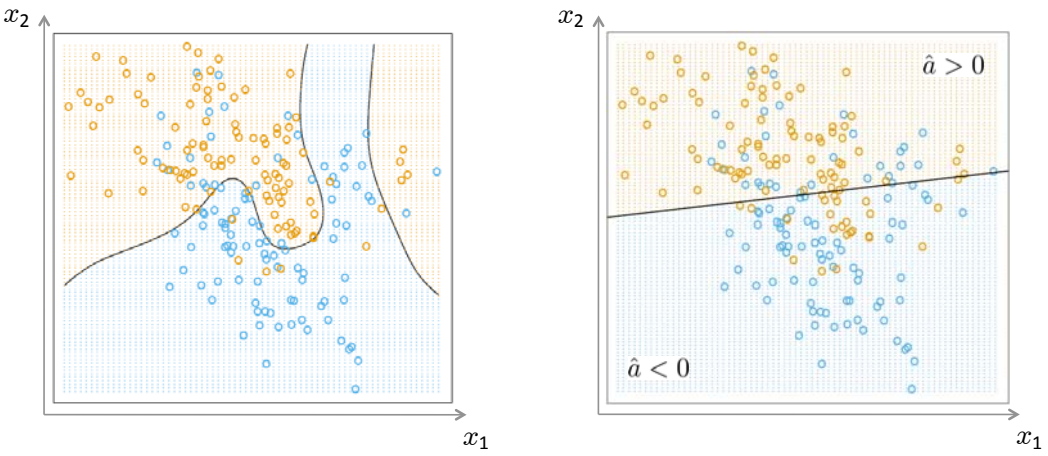


Figure 1. A two-dimensional classification problem. Left: Training data and optimal decision boundary calculated from the underlying distributions. Right: Fit of a linear discriminant function. (Figure adopted from [3], modified.)

2 (Re-)Inventing the neuron

While the simple linear approach outlined in the introduction already yields a decent solution to the problem of separating potential signal events from the background, we may ask the question: *"Given an event with values* $\mathbf{x}^{ev} \equiv (x_1^{ev}, x_2^{ev})$*, how reliable is our classification?"* Clearly, a quantitative measure of probability would be more desirable than a simple yes/no-decision. Obtaining such a measure is straightforward: we just have to map the unbounded co-domain of our function $\hat{a}$ onto the interval $[0, 1]$. A sigmoid transformation (eqn. 2) does the job.

$$\hat{a} \mapsto \sigma(\hat{a}) \equiv \frac{1}{1 + \exp(-\hat{a})} \tag{2}$$

Indeed, applying the sigmoid transformation offers a probabilistic interpretation of the classification problem, assigning values between 0 and 1 to the data points and mapping the decision boundary to 0.5 (equal probability for both signal and background) as expected. We may thus read the transformed output in the following way:

$$\sigma(\hat{a}(\mathbf{x})) \equiv p(signal|\mathbf{x}) \tag{3}$$

We have just invented the neuron! Indeed, in machine learning language a neuron is nothing but a linear combination of some input variables concatenated with a sigmoid transformation (figure 2). In this picture, our function $\hat{a}$ is called the *activation* of the neuron and $\sigma(\hat{a})$ is called its *activity*. Since we will mostly be concerned with the activity in the following, we will assign the symbol y to it from now on. Note that the entire behavior of the neuron is determined by the values of the parameters $\mathbf{w} \equiv (w_0, ..., w_i, ...w_N)$, which are called *weights*. The space of all possible weight configurations of a neuron is called the *weight space* (figure 3).

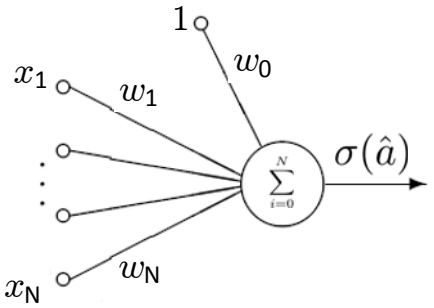


Figure 2. Visualization of a neuron: Weighted summation of the inputs and nonlinear (sigmoid) transformation at the output. (Figure adopted from [1], modified.)

Finding the neuron configuration which provides the best signal discrimination thus amounts to searching the weight space for the optimal set of weights for the training data set in question. This is called *training* or *learning* in the neural network jargon and is described in the following chapter.

3 Neuron training

As we have learned in chapter 1, training the neuron for a given classification task amounts to searching the optimal set of weights. We can tackle this problem with a more familiar looking maximum likelihood approach if we make use of the probabilistic interpretation of the neuron output. Remember that $y(\mathbf{x}) = p(signal|\mathbf{x})$, which implies $1 - y(\mathbf{x}) = p(background|\mathbf{x})$. Combining the two into an expression for the likelihood of the training data set $\mathcal{D}$ whose true class labels are known and taking the negative logarithm, we obtain the *error function* $E(\mathbf{w})$. It is given by

$$E(\mathbf{w}) = -\ln P(\mathcal{D}|\mathbf{w}) = \sum_n (t^{(n)} \ln p(C_1|\mathbf{x}^{(n)}) + (1 - t^{(n)}) p(C_0|\mathbf{x}^{(n)})) \tag{4}$$

where we have used the class label C_1 (C_0) to denote signal (background) and introduced the truth label t which takes the value 1 (0) for signal (background). The optimal weight configuration of

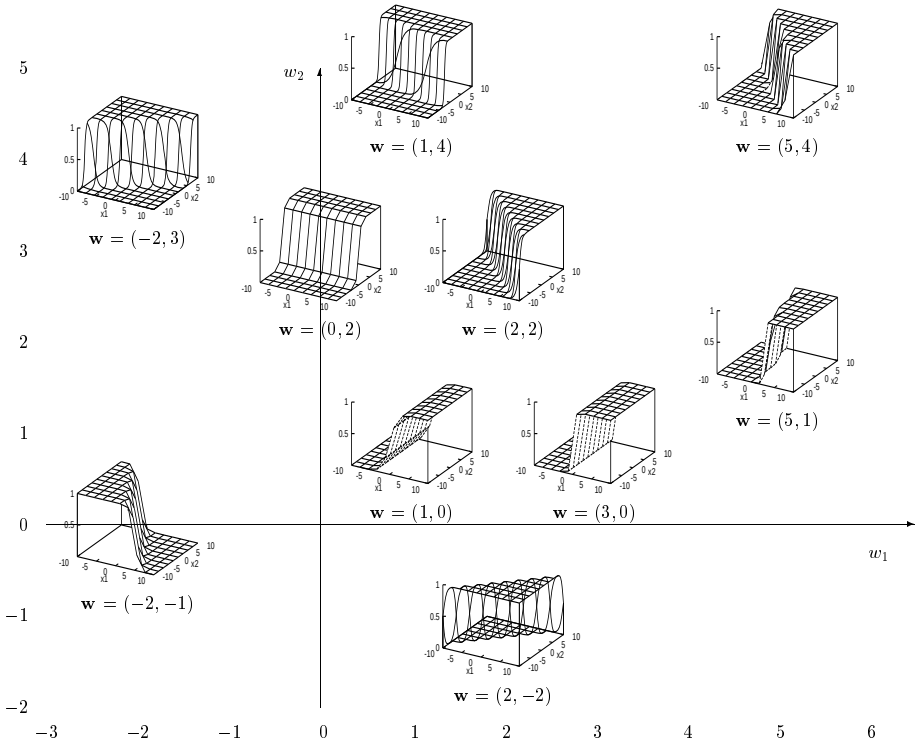


Figure 3. The *weight space* consists of all possible weight configurations of a neuron. Each one corresponds to a different discriminating function. (Figure adopted from [1].)

the neuron for a given training data set $\mathcal{D}$ can now be found by minimizing the error function. It is worthwhile to have a closer look at the minimization of the error function of the single neuron since we will rely on our insights when we discuss neural networks in chapter 4.

Although there are more sophisticated methods for minimization available, it is very instructive to consider a simple gradient descent approach for the error function. The idea is to move through the weight space by calculating the direction of steepest descent (which is just given by the negative gradient) and to take a step in that direction repeatedly. The update rule for the neuron weights thus reads:

$$\mathbf{w}^{(k)+1} = \mathbf{w}^{(k)} - \eta \frac{dE^{(k)}}{d\mathbf{w}} \tag{5}$$

The weights can either be updated separately for each event in the training data set (*online learning*) or after the entire training data set has been processed once (*batch learning*). A complete pass through the training data set is called an *epoch* and typically several epochs are needed to complete the training. Online learning is usually to be preferred in HEP applications since it speeds up the learning process for redundant data sets (like large MC samples with many similar events) and may help to avoid local minima by introducing a stochastic component to the minimization (see [5]). Leaving the

determination of the optimal step size η aside (for a discussion see [5]), we find that the direction of the step for a weight update is given by

$$\frac{dE}{dw_j} = \sum_n -(t^{(n)} - y^{(n)})x_j^{(n)}. \tag{6}$$

Note that this is proportional to the current output error $e^{(n)} = (t^{(n)} - y^{(n)})$ of the neuron which is the difference between the desired output (= the true class label) and the current output. This observation will later lead us to a simple scheme for the update rule in complex networks.

3.1 Overtraining in neurons

Although usually only an issue in more complex networks, we will now have a closer look at *overtraining* in the context of single neurons. This will lead to important insights on countermeasures that can also be applied in more complicated settings. Let's consider a simple training example with two variables. Since the training data set is small, it is sufficient to update the neuron weights after each epoch, i.e. use batch learning. The evolution of the weights (path in weight space) is shown in the first panel of figure 4. The other panels show subsequent updates of the decision boundary defined by the neuron during training. After a certain number of iterations, the weights start to evolve along a fixed direction in weight space, which means that the orientation of the decision boundary remains constant from this point on. However, the training does not stop here, but the weights keep growing, causing the slope of the neuron output function to get steeper and steeper as can be seen from the contours in figure 4.

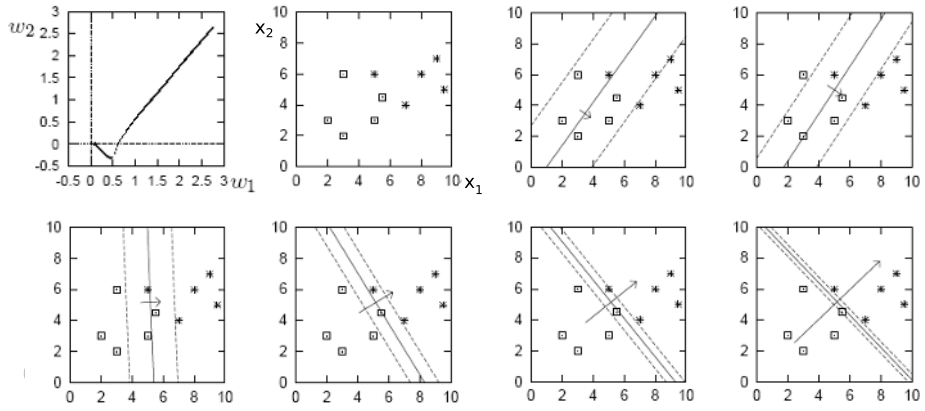


Figure 4. Evolution of the neuron weights and the decision boundary defined by the neuron during training. Overtraining occurs as the number of epochs during training gets large. The length of the arrow on the decision boundary is proportional to the gradient of the neuron output. (Figure adopted from [1], modified.)

This behavior is called *overtraining* and is undesirable because it means that the neuron will assign probabilities close to one (or close to zero) to any data point, even if that point is close to the decision boundary. Intuitively, one would expect that points close to the decision boundary get assigned almost

identical signal and background probabilities (that is, a neuron output of 0.5), but overtraining shrinks this "zone of undecidedness" and leads to overly confident predictions. Assessing the classification performance of such an overtrained neuron on an independent test data set will usually yield a bad result, because the neuron may not only assign data points close to the decision boundary to the wrong class, but it will do so assigning very large (or very small) signal probabilities, resulting in a large output error.

3.1.1 Avoiding overtraining

There are several ways to avoid overtraining. An obvious and simple recipe would be to stop the training after a fixed number of iterations, interrupting the learning process before the weights can diverge ("early stopping"). But how should one determine the optimal point to stop? And how does one ensure that the training has already converged to the best set of weights? Clearly, a more informed approach is called for.
We have already briefly mentioned that the performance of a given classifier can be assessed on an independent test data set. Since overtraining leads to a degradation of the performance on this data set, one can in principle monitor the classification performance of the neuron on the test data set during training and stop when the misclassification error reaches its minimum on the test data. (Note that the error function measured on the training data set will always keep decreasing and thus does not provide insights into overtraining). Despite being theoretically well founded, this approach is time consuming due to the extra evaluation of the test data set. It also requires a large amount of Monte Carlo data for two reasons: First, a test data set is definitely needed, and second, an unbiased performance assessment now calls for a third independent set, since the test data set has already been used in the optimization process.

It turns out that we can avoid this overhead by introducing the concept of *regularization*. This refers to the idea that the weights of the neuron are regularized by introducing a penalty for large weights. This can for example be realized by adding a term proportional to the sum of squared weights (often referred to as a *weight decay* term) to the error function:

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\alpha}{2} \sum_i w_i^2 \tag{7}$$

Let's have a look at the behavior of the regularized neuron during training. Using the same training data as before, we now obtain the results displayed in figure 5. The weights are now bounded and the contours of the decision boundary remain unchanged after a certain number of iterations which is determined by the size of the regularization parameter α. The resulting effect is similar to what can be achieved by early stopping. At first glance, one might thus say that we have just shifted the problem from making an arbitrary choice for the number of iterations to making an arbitrary choice for α. But when we return to the discussion of regularization in neural networks later on, we will see that the optimal choice of the regularization parameter can be inferred from the training data itself in what is called the *evidence framework*.

4 From neurons to networks

Having discussed the single neuron thoroughly in the first chapter, the generalization to neural networks composed from several neurons is now straightforward. But how should the neurons be arranged to form a neural network? Let me remind you of the universal approximation theorem, which forms the theoretical foundation for the modeling of functions through neural networks. It reads:

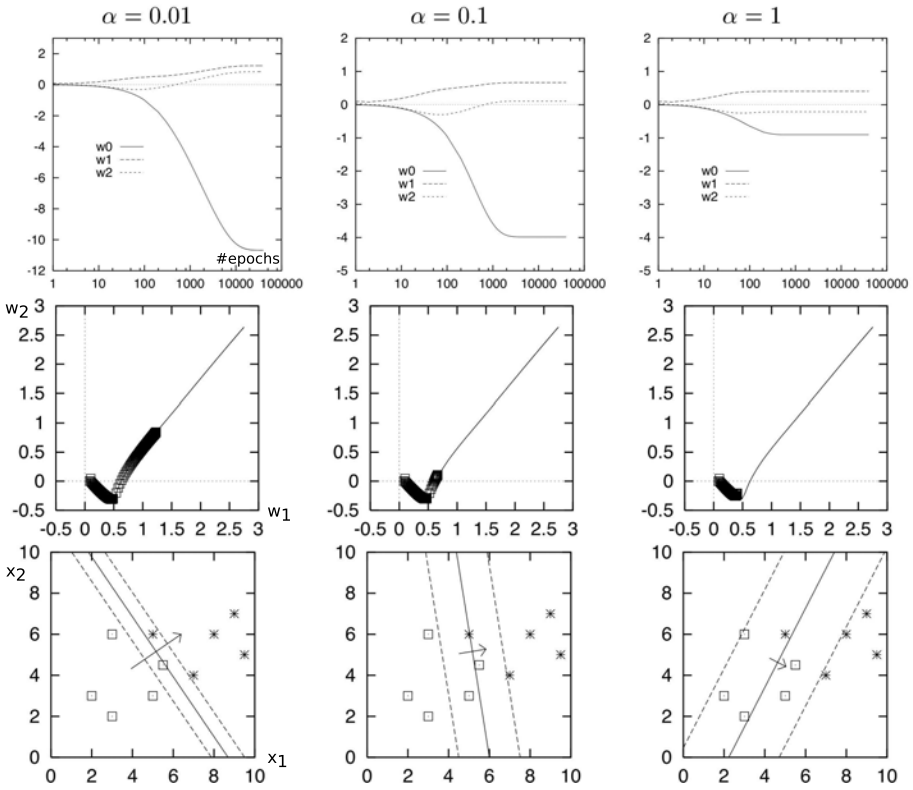


Figure 5. Effect of regularization for different values of the regularization parameter α on the evolution of neuron weights during training. Top row: The weights are bounded and stay constant after a certain number of iterations. Middle row: Weight evolution now follows the path marked by the black squares - a behavior similar to early stopping is observed. Bottom row: The decision boundary defined by the neuron after convergence of the weights. (Figure adopted from [1], modified.)

Let $\sigma(\cdot)$ be a non-constant, bounded and monotone-increasing continuous function. Let $C(I_D)$ denote the space of continuous functions on the D-dimensional hypercube. Then, for any given function $f \in C(I_D)$ and $\epsilon > 0$ there exist an integer M and sets of real constants w_j, w_{ji} where $i = 1, ..., D$ and $j = 1, ..., M$ such that

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_j \sigma \left(\sum_{i=1}^{D} w_{ji} x_i + w_{j0} \right) \tag{8}$$

is an approximation of $f(\cdot)$, that is $|y(\mathbf{x}) - f(\mathbf{x})| < \epsilon$.

The notation was not chosen arbitrarily here - identifying the parameters w_j, w_{ji} with the weights and $\sigma(\cdot)$ with the sigmoid transform, a sloppy translation of the theorem reads:

One can build any continuous function from neurons!

But the theorem not only tells us that we can approximate any function with neurons, but also how to do it: Figure 6 shows that the structure described by formula 8 is composed of neurons organized in layers, where the output of a neuron in one layer becomes an input to all the neurons in the following layer. This structure is called a *feed-forward network* : The first layer provides connections for all input variables and passes them on to an intermediate layer of neurons, where they are summed and transformed (innermost term in formula 8). Subsequent layers may follow until the network response is provided at the output of the neurons in the last layer (*output layer*). This last layer will often be linear, so that the entire non-linearity (and hence the flexibility to approximate arbitrary functions) of the network is provided by the intermediate layers. These layers are called *hidden layers* and the neurons they consist of are referred to as *hidden neurons* or *hidden units* because they are not directly connected to the network input or output. Different network architectures are often classified by the number of hidden layers and the term *single layer network* is often used to describe a network with one hidden layer only.

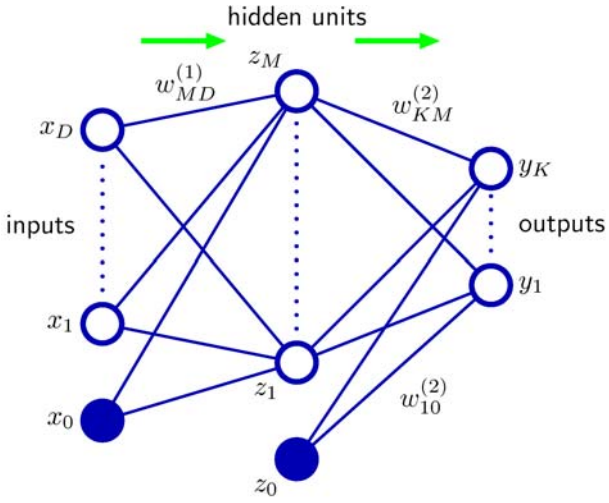


Figure 6. A feed-forward neural network consists of neurons organized in layers. Neurons in one layer provide the input for the neurons in the following layer. (Figure adopted from [4], modified.)

4.1 Network complexity

Figure 7 illustrates the approximation of several simple functions by a single layer network with three hidden units. There is only one input variable and the network output is one-dimensional. The dashed lines represent the output values of the hidden neurons which are then linearly combined in the output layer to form the network output. Since the approximation of functions by such a network resembles an expansion of the target function in a basis consisting of sigmoid functions (=neurons), it is intuitively clear that the number and complexity of the functions that a network can represent grows

with the number of hidden neurons. But the complexity is not only determined by the number of hidden neurons, but also by the typical size of the weights **w** throughout the network. In our discussion of neuron training we have seen that large weights correspond to a steep decision boundary, that is a large variation of the neuron output over a small interval of the input values. The approximation of functions with intricate small scale features must thus contain neurons with large weights which provide a sufficient change of the output value in a small interval. It can even be shown that in the limit of infinite networks ($n_{neuron} \to \infty$) the complexity of the functions that can be represented is entirely determined by the characteristic size of the weights [9]. In our discussion of network training we will thus assume that the network in question is always large enough to represent the discriminating function we are interested in and that the training task amounts to finding the optimal set of weights.

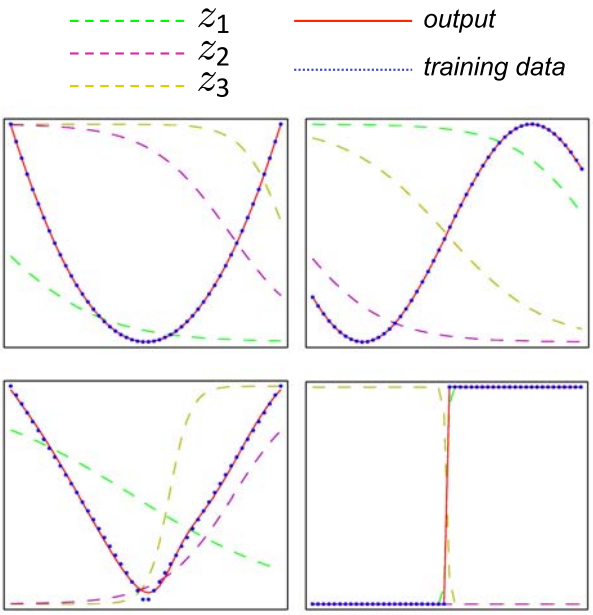


Figure 7. Approximation of simple functions by a single layer network with three hidden units. The outputs of the hidden units are labeled z_1, z_2, z_3. (Figure adopted from [4], modified.)

4.2 Network training

Recalling what we have learned from our discussion of neuron training, we know that the best configuration of the weights can be found by following the gradient of the error function through the weight space until we arrive at a minimum. But deriving an update rule for all the weights in a large network seems to be very difficult since a lot of complicated derivatives are involved. Luckily, Rumelhart et al. have shown that the update rules for any network can be easily derived by an algorithm known as *backpropagation* [6]. I will not give a detailed description of backpropagation here, but just briefly outline the rules that can be derived from it. First recall that the weights of a single neuron are updated by taking a step along the gradient of the error function which is proportional to the current size of the output error in each coordinate direction:

$$\frac{dE}{dw_k} = (y - t)x_k \tag{9}$$

It turns out that a very similar rule applies to the neurons in a network:

$$\frac{dE}{dw_{ij}} = \delta_j z_i \tag{10}$$

$$\text{where } \delta_k = (y_k - t_k) \text{ for output neurons} \tag{11}$$

$$\text{and } \delta_j \propto \sum_k w_{kj}\delta_k \text{ else.} \tag{12}$$

Careful examination of this rule shows that a neuron in the output layer behaves in exactly the same way as a single neuron: The update for a given weight is obtained by multiplying the input associated with that weight (which is one of the outputs from the previous layer) by the current output error $e_k \equiv \delta_k = (y_k - t_k)$ of that neuron. Note that the updates in the other layers can then be performed subsequently by propagating the output error "backwards" through the network according to equation 12 - hence the name "backpropagation".

We have just discovered one of the characteristic rules of network training: While input information is passed forward from layer to layer, errors are passed backward in the same way during training (figure 8). This allows for very efficient network implementations which exploit the forward-backward symmetry to speed up the training.

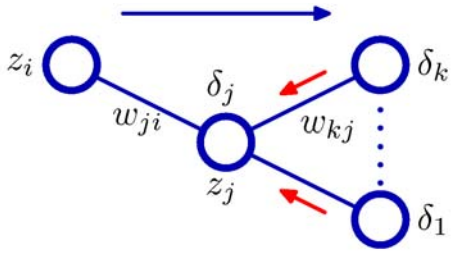


Figure 8. Backpropagation of errors in a neural network. (Figure adopted from [4].)

4.3 Overtraining revisited

We have seen that a network with a sufficient number of hidden neurons can evolve into a representation of the optimal discriminating function for a given problem by adapting to the training data set during the learning phase - but what if the network adapts *too well* to the training data? This situation may be compared to a fitting problem where one tries to describe a set of data points which where generated from a linear model with some noise on top (the noise is usually associated with an uncertainty of the measurement) by polynomials of varying order. Even though the underlying model is linear, a polynomial of sufficient order will always be able to pass through all the data points and thus produce smaller residuals. But this supposedly better fit will have low predictive power since it does not generalize well: An interpolation (or even more extreme an extrapolation) using this polynomial will yield predictions which typically differ significantly from the underlying model. *Overfitting* has occurred. In the realm of neural networks, a behavior like this is called *overtraining*, and we have already encountered it in our discussion of neuron training, where it lead to overly confident class assignments.

In complex networks, overtraining can be as extreme as "learning the class labels of the training data set" and lead to a severe deterioration of the classification performance on any other sample. The functions represented by such a heavily overtrained network will usually display numerous small

scale features which correspond to the statistical fluctuations found in the training data (see figure 9). Just as a polynomial of high order will give poor results when interpolating a linear model, a network like this will not generalize well to data not encountered during training.

4.4 Regularization in neural networks

How can we avoid overtraining in neural networks? One may of course reduce the size of the network, thus limiting the complexity of functions representable by the network. But as the type of the optimal discriminating function is usually unknown, one might unwantedly exclude it from the set of accessible functions this way. Stopping the training prematurely as discussed earlier in the context of neuron training is also discouraged: Undesirable small scale features may already develop in certain regions of the phase space before a satisfactory global solution has been found. The observation made earlier, stating that intricate small scale variations of the function represented by the network indicate the presence of large weights, provides a strong hint that weight regularization may be effectively used to reduce overtraining and produce smooth and well generalizing decision boundaries. Adding the quadratic regularizer term encountered earlier to the error function of the network and choosing an appropriate regularization strength parameter α indeed leads to a well behaved solution to our initial classification problem displayed in figure 9. Not only do unwanted statistically induced features of the original (unregularized) discriminating function (like disconnected regions in which data will be assigned to the same class) vanish, but also does the regularized function follow the true decision boundary as induced from the underlying distributions more closely. An inspection of the network weights would show that this behavior can indeed be attributed to a reduction of the average weight size. One nagging question remains, though: How can we determine the optimal amount of regularization? Is there a well informed approach to set α or are we left with heuristics only? To answer this question, we have to see how neural networks can be understood from the viewpoint of Bayesian statistics - a field which seems utterly unrelated at first glance.

5 Joining Bayesian statistics and neural networks

Bringing together Bayesian statistics and neural networks looks like a strange idea in the first place - an approach guided by the fundamental principles of probability and a field of statistical learning largely governed by heuristics don't seem to go together too well. Nevertheless we will see that it is this connection which sheds light on the problem of optimal regularization.
Remember that the neuron output y can be interpreted as a probability for each event and that we can express this probability using the truth class label $t \in \{0, 1\}$:

$$P(t = 1|\mathbf{w}, \mathbf{x}) = y \tag{13}$$

$$P(t = 0|\mathbf{w}, \mathbf{x}) = 1 - y \tag{14}$$

$$\Rightarrow P(t|\mathbf{w}, \mathbf{x}) = y^t(1 - y)^{1-t} = \exp[t \ln y + (1 - t) \ln(1 - y)] \tag{15}$$

Note that the exponent is just the error function we introduced in section 3 which allows us to write

$$P(\mathcal{D}|\mathbf{w}) = \exp(-E(\mathbf{w})). \tag{16}$$

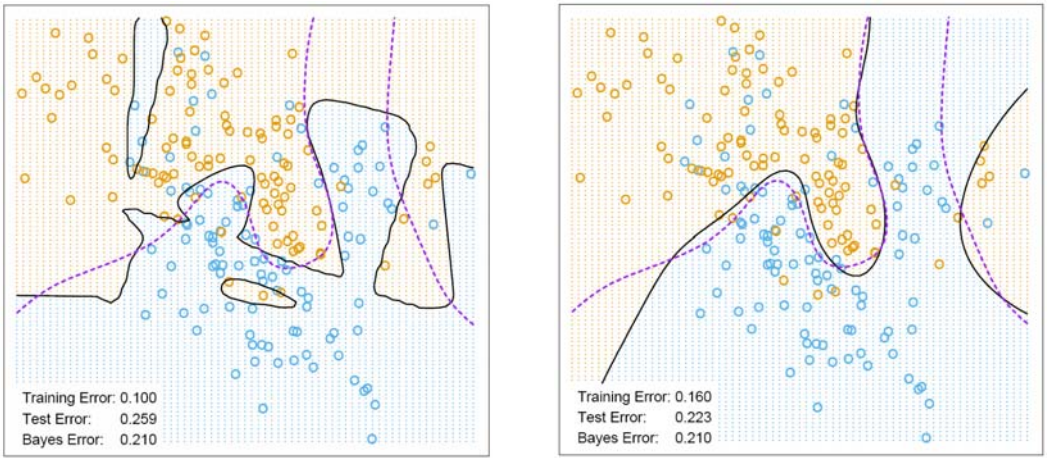


Figure 9. Application of a single layer network with ten hidden units to the classification problem from section 1. Left: Overtraining is clearly visible, the error on the training data set is much smaller than the test error. Right: Regularization smooths the decision boundary and improves the performance on the test data. The Bayes error (measure of best achievable performance) is given for comparison. (Figure adopted from [3].)

If we try a similar interpretation for the regularization (weight decay) term, we see that it resembles a Gaussian log-probability distribution for $\mathbf{w}$:

$$P(\mathbf{w}|\alpha) = \frac{1}{Z_{WD}(\alpha)} \exp(-\frac{\alpha}{2}(\mathbf{w}^\top \mathbf{w})) \tag{17}$$

where $Z_{WD}(\alpha)$ is an appropriate normalization constant.
We can now combine the two using Bayes' theorem to obtain an expression for the probability distribution of the networks parameters $\mathbf{w}$ for a given training data set $\mathcal{D}$ and regularization strength α. The likelihood is now represented by the error function while the prior is represented by the regularization term.

$$P(\mathbf{w}|\mathcal{D}, \alpha) = \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w}|\alpha)}{\int P(\mathcal{D}|\mathbf{w})P(\mathbf{w}|\alpha)d\mathbf{w}} = \frac{1}{Z_{\tilde{E}}} \exp(-\tilde{E}(\mathbf{w})) \tag{18}$$

With this insight we can reinterpret the training of a neural network as an inference task where the optimal set of weights maximizes the posterior probability defined in equation 18. This can be illustrated by returning to the simplest possible network - a single neuron - and following the evolution of the posterior distribution of the neuron weights $P(\mathbf{w}|\mathcal{D}, \alpha)$ during training as more and more data is considered (figure 10).

5.1 Network regularization in the evidence framework

Let's return to the problem of optimal weight regularization. Making use of our new probabilistic interpretation of network training, this amounts to finding an expression for the posterior distribution of α:

$$P(\alpha|\mathcal{D}) \propto P(\mathcal{D}|\alpha)P(\alpha). \tag{19}$$

If we assume no prior knowledge about the regularization parameter α (e.g. a flat prior $P(\alpha)$), Bayes' theorem implies that $P(\alpha|\mathcal{D})$ is proportional to $P(\mathcal{D}|\alpha)$, which can be written as

$$P(\mathcal{D}|\alpha) = \int P(\mathcal{D}|\mathbf{w})P(\mathbf{w}|\alpha)d\mathbf{w}. \tag{20}$$

Close inspection of equation 18 reveals that this integral corresponds to the normalization constant $Z_{\tilde{E}}$ of the posterior distribution of $\mathbf{w}$, which is often called the *evidence*. The optimal regularization strength can thus be inferred from the training data by maximizing the evidence with respect to α. Unfortunately, the integral in equation 20 is analytically intractable, thus we have to resort to a Laplace approximation.

5.1.1 Evaluating the evidence

Consider a smooth function $f(\mathbf{w})$ of the network parameters $\mathbf{w}$, which has a maximum at the most probable value $\mathbf{w}_{MP}$. Obviously, also $\ln f(\mathbf{w})$ has a maximum at $\mathbf{w}_{MP}$ and Taylor expanding around that point gives

$$\begin{aligned} \ln f(\mathbf{w}) &\approx \ln f(\mathbf{w}_{MP}) - \frac{1}{2}(\mathbf{w}-\mathbf{w}_{MP})^\top \mathbf{A}(\mathbf{w}-\mathbf{w}_{MP}) \\ \text{with } \mathbf{A} &= -\nabla\nabla \ln f(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_{MP}}. \end{aligned} \tag{21}$$

Exponentiating this expression we obtain a Gaussian approximation of $f(\mathbf{w})$ around the maximum:

$$f(\mathbf{w}) \approx f(\mathbf{w}_{MP}) \exp(-\frac{1}{2}(\mathbf{w}-\mathbf{w}_{MP})^\top \mathbf{A}(\mathbf{w}-\mathbf{w}_{MP})) \tag{22}$$

We can now apply this approximation to the evidence $Z_{\tilde{E}}$ and solve the resulting Gaussian integral to obtain:

$$\begin{aligned} P(\mathcal{D}|\alpha) &= \int P(\mathcal{D}|\mathbf{w})P(\mathbf{w}|\alpha)d\mathbf{w} \\ &= \int \exp(-E(\mathbf{w}))\cdot(\frac{\alpha}{2\pi})^{W/2}\exp(-\frac{\alpha}{2}\mathbf{w}^\top\mathbf{w})d\mathbf{w} \\ &= (\frac{\alpha}{2\pi})^{W/2}\exp(-E(\mathbf{w}_{MP}) - \frac{\alpha}{2}\mathbf{w}_{MP}^\top\mathbf{w}_{MP}) \\ &\quad \cdot \int \exp(-\frac{1}{2}(\mathbf{w}-\mathbf{w}_{MP})^\top\mathbf{A}(\mathbf{w}-\mathbf{w}_{MP}))d\mathbf{w} \\ &= \frac{\alpha^{W/2}}{(\det\mathbf{A})^{1/2}}\exp(-\tilde{E}(\mathbf{w}_{MP})) \\ \Rightarrow \ln p(\mathcal{D}|\alpha) &= -\tilde{E}(\mathbf{w}_{MP}) + \frac{W}{2}\ln\alpha - \frac{1}{2}\ln(\det\mathbf{A}) \end{aligned} \tag{23}$$

The second and third term are called *Occam terms*. They balance the contribution from the error function in such a way that the resulting model complexity will provide both a good fit to the training data set and a high generalizability. A schematic view of the connection between $\ln p(\mathcal{D}|\alpha)$ and the classification performance on the training/test sample is depicted in figure 12. Figure 11 shows an example for evidence-based regularization in a two dimensional classification problem.

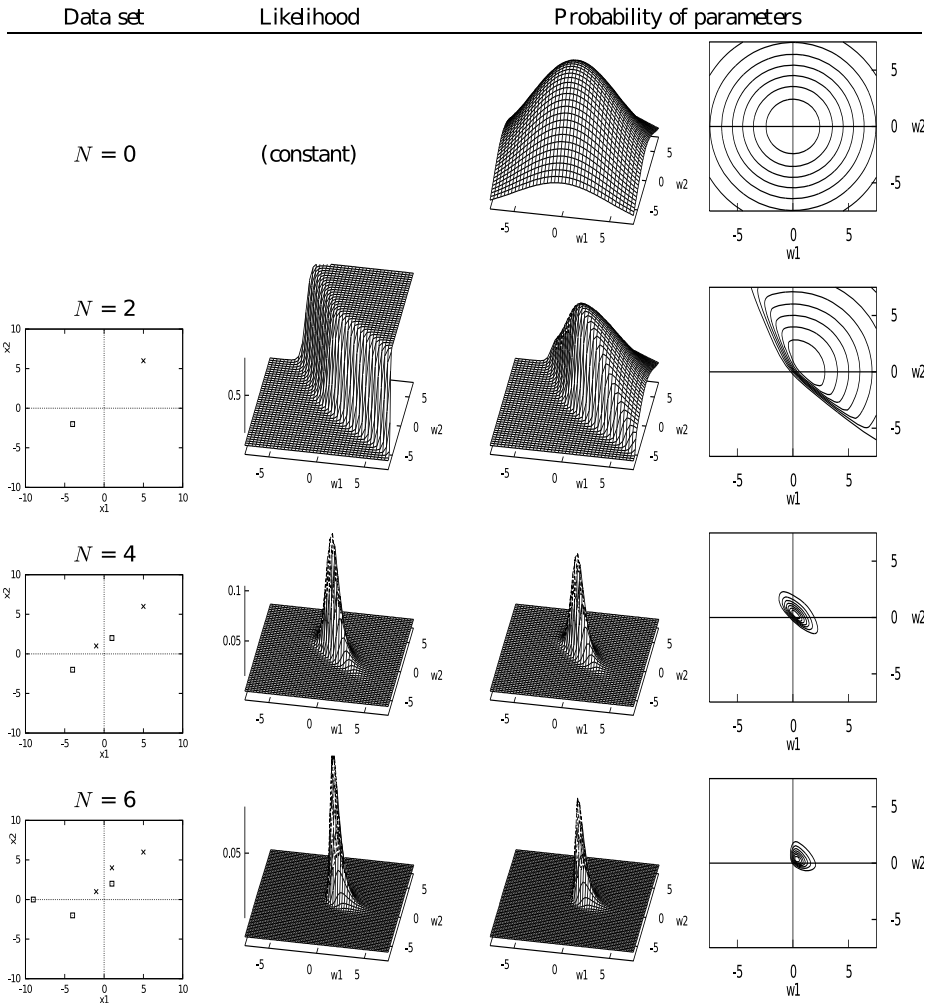


Figure 10. Evolution of the posterior distribution of the weights during the training of a neuron. (Figure adopted from [1]).

5.1.2 Variable selection through regularization

In high energy physics analyses, the selection of discriminating variables is usually motivated by the features of the underlying physics process. But in many situations it is not obvious which variables will provide the best discrimination power, in particular when they are combined in a multivariate method. Surprisingly, evidence-based network regularization can also be used to prune useless variables from a model and thus help to identify the relevant ones for a given problem. This is achieved by generalizing the weight-decay approach by introducing separate regularization constants α_i for all input variables of the network (see figure 13). The connection weights for different input nodes can now be regularized individually and the weights associated with uninformative variables tend to be

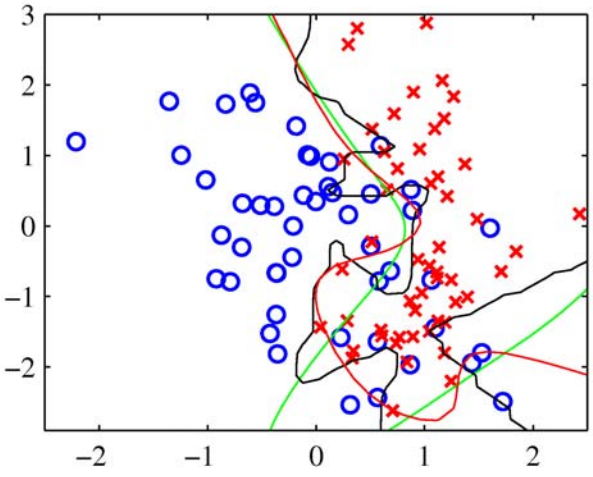
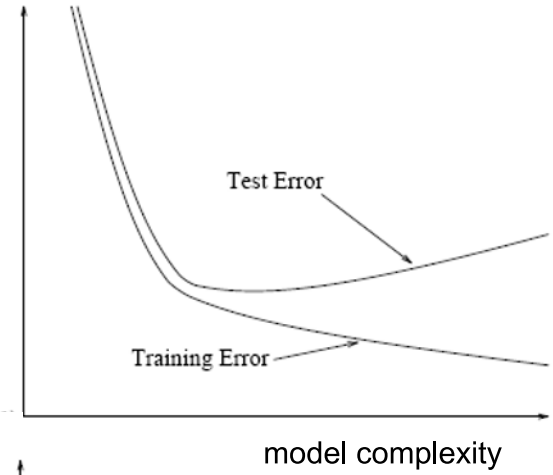


Figure 11. Application of evidence-based regularization to a single layer network with eight hidden units. Green: Optimal decision boundary. Black: Decision boundary obtained with unregularized training. Red: Decision boundary obtained with evidence-based regularization. (Figure adopted from [4].)



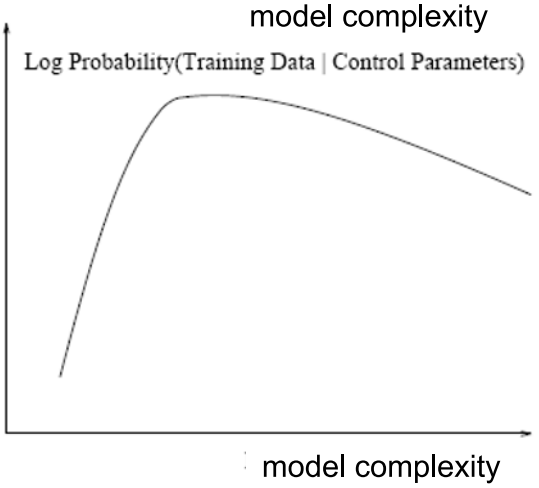


Figure 12. Relation between model complexity, evidence and classifier test performance. (Figure adopted from [2], modified.)

constraint to values close to zero which means that they are effectively dropped from the model. This method is known as *Automatic Relevance Determination (ARD)* (MacKay and Neal, unpublished), see also [2].

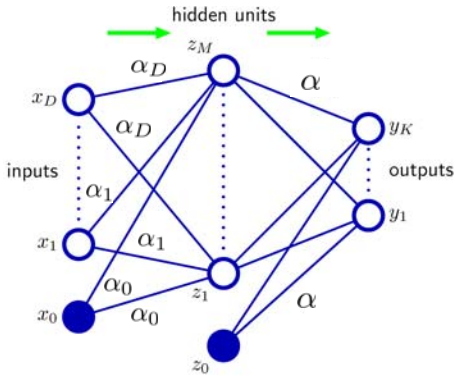


Figure 13. A neural network with separate regularization parameters for each input variable - a configuration used for *Automatic Relevance Determination*. (Figure adopted from [4], modified.)

5.2 Predictions and confidence

The Bayesian approach to neural networks also provides a natural way to embody the uncertainty of the network output. When the fully trained network is applied to data that was not encountered during training to make predictions, its output $P(t^{N+1}|\mathbf{x}^{N+1})$ is usually calculated using just the most probable set of weights derived during training. A look at our initial neuron training example may suffice to convince ourselves that this approach is not optimal: Consider two new data points **A** and **B** (left panel of figure 14). Not only will they be assigned to the same class, but they will also be assigned the same probability since they share the same distance from the decision boundary. Wouldn't it be more natural if the class prediction for **B** was less confident since the training data is scarce and thus the model description probably poor in that region? The reason for that shortcoming is that we have implicitly approximated the posterior distribution of the weights by a delta-distribution at $\mathbf{w}_{MP}$, neglecting the spread of $P(\mathbf{w}|\mathcal{D},\alpha)$ and thus the uncertainty associated with the weights:

$$\begin{aligned} P(\mathbf{w}|\mathcal{D},\alpha) &\approx \delta(\mathbf{w}-\mathbf{w}_{MP}) \\ \Rightarrow P(t^{N+1}|\mathbf{x}^{N+1},\mathcal{D},\alpha) &= \int P(t^{N+1}|\mathbf{x}^{N+1},\mathbf{w})\delta(\mathbf{w}-\mathbf{w}_{MP})d\mathbf{w} \\ &= P(t^{N+1}|\mathbf{x}^{N+1},\mathbf{w}_{MP}) \end{aligned} \tag{24}$$

If we drop the delta-approximation, we face another analytically intractable integral in equation 24 - but as seen before we can resort to a Gaussian approximation to obtain an estimate of the predictive posterior $P(t^{N+1}|\mathbf{x}^{N+1},\mathcal{D},\alpha)$. Using this improved method to make predictions yields the result shown in the right panel of figure 14: The farther one moves in the phase space from the bulk of the training data, the farther the contours spread from the decision boundary. The prediction for point **B** has a greater uncertainty than that for point **A**. The application of that principle to a neural network with eight hidden units is shown in figure 15.

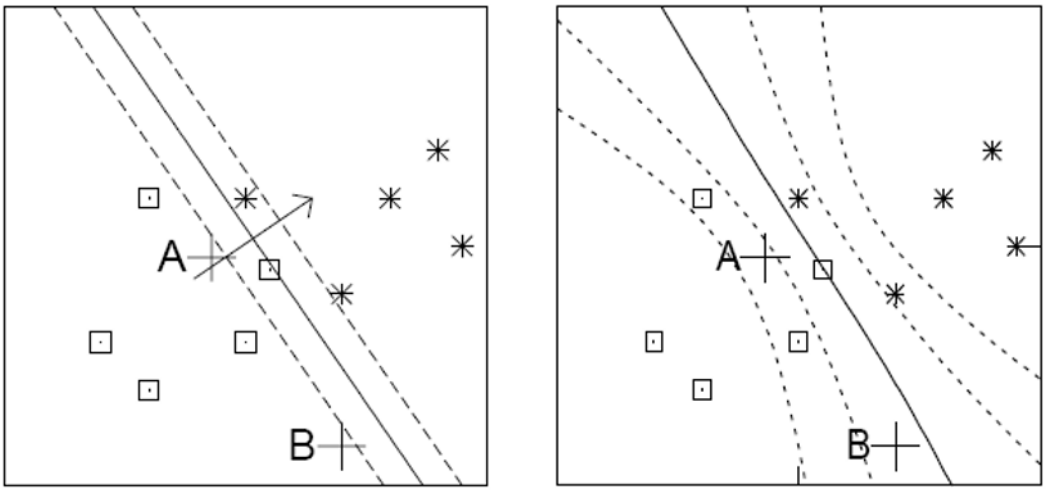


Figure 14. Contours of the decision boundary as defined by a fully trained neuron. Left: Approximation of the posterior distribution of the neuron weights by a delta-distribution. Right: Evaluation of the expectation by Gaussian approximation. (Figure adopted from [1].)

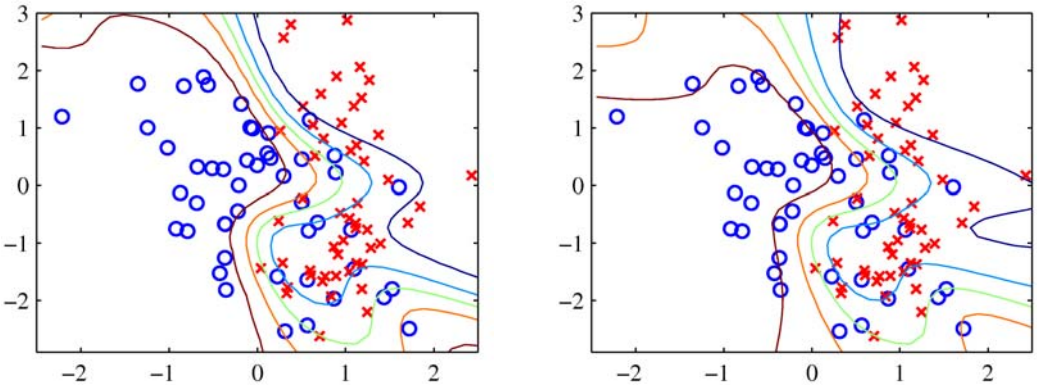


Figure 15. Contours of the discrimination function defined by a neural network with eight hidden units. Left: Approximation of the posterior of the weights by a delta-distribution. Right: Gaussian approximation of the posterior. (Figure adopted from [4].)

6 Conclusions

Neural networks are no black boxes - we have seen how a neuron can be constructed by transforming a linear discriminant function into a probability statement and how the universal approximation theorem dictates how neurons can be combined to form universal function approximators known as feed-forward neural networks.

The Bayesian view of neural network training as an inference task for the weights has led to interesting insights on network regularization and the uncertainty of network predictions.

7 Further reading and neural network software

Most of the figures in this text were taken from the books of MacKay [1], Bishop [4], and Hastie, Tibshirani and Friedman [3]. These books provide an excellent resource for further reading on neural networks. A lot of useful practical hints for users of neural networks are given in [5].
Among the neural network software packages most widely used in high energy physics are TMVA [7] and NeuroBayes [8].

8 Acknowledgments

I would like to thank the organizers for giving me the opportunity to give a lecture at the school of statistics. The program of the school was diverse and I enjoyed many interesting discussions. I would also like to thank the authors of the aforementioned books for the permission to use their illustrative figures for this introductory text.

References

[1] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003, www.inference.phy.cam.ac.uk/mackay/itila/

[2] D.J.C. MacKay, "Bayesian Methods for Neural Networks: Theory and Application, *Course notes for Neural Networks Summer School*, 1995, http://www.inference.phy.cam.ac.uk/mackay/BayesNets.html

[3] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, Springer, 2nd edition, 2009

[4] C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006

[5] Y. LeCun, L. Bottou, G.B. Orr, K.R. Mueller, "Efficient BackProp", in *Neural Networks: Tricks of the trade*, Springer, 1998

[6] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning representations by back-propagating errors", in *Nature*, 323 533-536

[7] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, and H. Voss, "TMVA: Toolkit for Multivariate Data Analysis", PoS A CAT 040 (2007) [physics/0703039].

[8] M. Feindt, U. Kerzel, "The NeuroBayes neural network package", *Nuclear Instruments and Methods in Physics Research*, 2006, Vol. 559 Issue 1, 190-194

[9] R.M. Neal, "Priors for infinite networks", Technical Report CRG-TR-94-1, Dept. of Computer Science, University of Toronto, 1994