# Artificial neural networks - Exercise session 2

Bayesian learning in neural networks

2015-2016

## 1 Understanding Bayesian Inference: the Case of Network Weights

For simplicity of exposition, we begin by considering the training of a network for which the architecture is fixed in advance. More precisely we focus on the case of a one-neuron network. In the absence of any data, the distribution over weight values is described by a prior distribution denoted $p(w)$, where $w$ is the vector of adaptive weights (normally also biases, but in our example we will consider the case without bias). We also denote by $D$ the available dataset. Once we observe the data, we can write the expression for the posterior probability distribution of the weights using Bayes theorem:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \tag{1}$$

where $p(D)$ is a normalization factor ensuring that $p(w|D)$ gives unity when integrated over the whole weight space. $p(D|w)$ corresponds to the likelihood function used in maximum likelihood techniques.
Since in the beginning we do not know anything about the data, the prior distribution of weights is set to (for example) a Gaussian distribution. The expression of the prior is then:

$$p(w) = (\frac{\alpha}{2\pi})^{\frac{W}{2}} \exp(-\frac{\alpha}{2}||w||^2) \tag{2}$$

with $W$ the number of weights, and $\alpha$ is the inverse of the variance. For simplicity we will choose $\alpha = 1$.
This choice of the prior distribution encourages weights to be small rather than large, which is a requirement for achieving smooth network mappings. So when $||w||$ is large, the parameter of the exponential is large, and thus $p(w)$ is small (small probability that this is the correct choice of weights. Things are reversed for small $||w|$).

**A concrete example: binary classification** We consider the case where input vectors are 2-dimensional $x = (x_1, x_2)$, and we have four data points in our dataset as in figure 1. We take a network of a single neuron (compare to the perceptron of the first exercise session), so there is only a single layer of weights, and choose the logistic function as transfer function:

$$y(x; w) = \frac{1}{1 + \exp(-w^T x)} \tag{3}$$

The weight vector $w = (w_1, w_2)$ is two-dimensional and there is no bias parameter. We choose a Gaussian prior distribution for the weights (with $\alpha = 1$). This prior distribution is plotted in figure 2.
The data points can belong to one of the two classes (cross or circle), the output $y$ giving the membership to one of these classes. The likelihood function $p(D|w)$ in Bayes theorem will be given by a product of factors, one for each data point, where each factor is either $y$ or $(1 - y)$ according to whether the data point belongs to the first or the second class.

First we consider just the points labeled (1) and (2). Then we consider all four points and recompute the posterior distribution of the weights.

Note: For an example of this case, run demo `bayesNN`, that you can find in the Exercise session 2 folder on Toledo → Assignments .

After training with only the first two data points, we see that the network function is a sigmoidal ridge ($w_1$ and $w_2$ control

the orientation and the slope of this sigmoid). Weight vectors from approximately half of the weight space will have probabilities close to zero, as they represent decision boundaries with wrong orientation.When using all 4 data points, there is no boundary to classify all $4$ points correctly. The most probable solution is the one corresponding to the peak point of the sigmoid, the others having quite low probabilities (so the posterior distribution of the weights is relatively narrow).

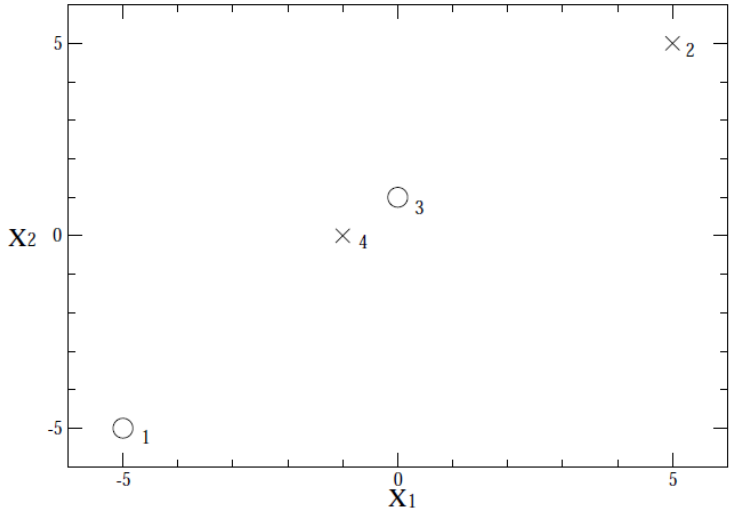


Figure 1: The four numbered data points in the dataset.

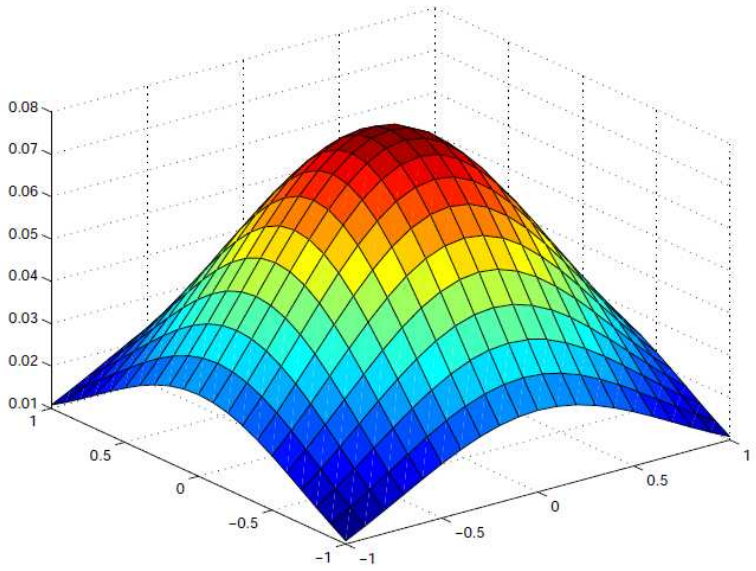


Figure 2: The prior distribution: a Gaussian.

## 2 Bayesian Inference of Networks hyperparameters

From an optimization point of view the training is performed by iteratively adjusting $w$ so as to minimize an objective function $M(w)$. In a maximum likelihood framework, $M$ is taken to be the error function $E_D(w)$ that depends on the data $D$. A common prescription to avoid overfitting is to include in $M$ a regularization term $E_W(w)$ (a.k.a. weight decay) to favor small values of

the parameter vectors, as discussed above. In particular, if we let

$$M(w) = \beta E_D(w) + \alpha E_W(w) \tag{4}$$

we can immediately give to this a Bayesian interpretation. In fact it is not difficult to show that $\beta E_D$ can be understood as minus the log likelihood for a noise model whereas $\alpha E_W$ can be understood as minus the log prior on the parameter vector. Correspondingly, the process of finding the optimal weight vector minimizing $M$ can be interpreted as a Maximum a Posteriori (MAP) estimation. Notice that we have implicitly considered $\alpha$ and $\beta$ fixed here. However these are hyperparameters that control the complexity of the model. Once more within a Bayesian framework one can apply the rules of probability and find these parameters accordingly. In MATLAB for a general feedforward neural network this can be accomplished by means of `trainbr`. This training function updates the weight according to Levenberg-Marquardt optimization. It minimizes (4) with respect to $w$ and determines at the same time the correct combination of the two terms so as to produce a network that generalizes well.

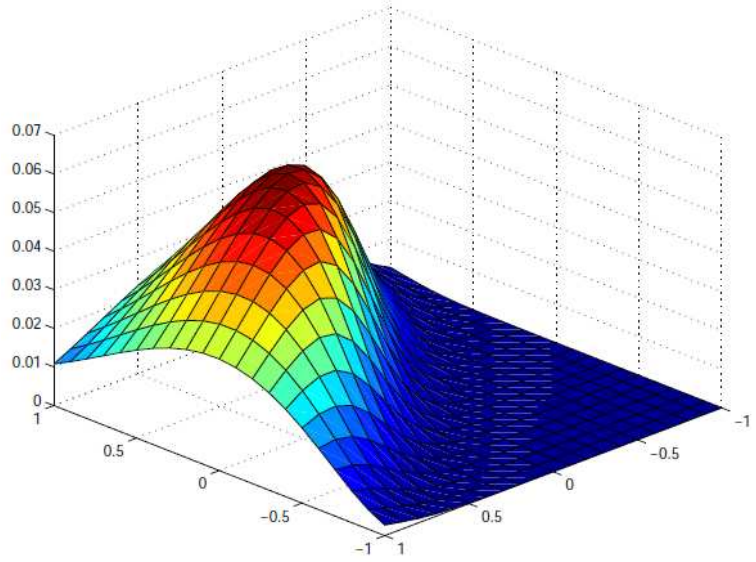


Figure 3: The distribution after presenting all four data points. Half of the weight space has become very improbable.

# 3 Exercises

- Run the `bayesNN` demo and understand it.
- Create a perceptron with one neuron, two inputs, no bias and arbitrary weights. Use this perceptron to generate input/output pairs with inputs arbitrary in $[-1, 1]$. Use these input/output pairs one by one to update a prior weight distribution. Plot the final most probable decision boundary. Is it close to the boundary of the perceptron? Is the weight of the perceptron close to the maximum of the posterior distribution? Use the demo `perbayes` as an example (`playshow perbayes`)
- Use `trainbr` to analyse the artificial datasets of Exercise Session 1, section 2 (you can use `bayesian_training` as a template) and compare it with the training functions seen during the first exercise session. Compare the test errors. Consider overparametrized networks (many neurons): do you see any improvement with `trainbr`?

# 4 Report

Write a report about the exercises of section 3. The report should consist of maximum 2 pages (including text + figures) to discuss the results of applying `trainbr` on the datasets used in the first Exercise Session (section 2).

# References

[1] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press.

[2] Matlab scripts for the exercises available in Toledo.

[3] Lecture slides and references therein.