Report 2: Bayesian learning in neural networks

Author: *Jannes Nys*

In this report, I discuss how Bayesian learning can be used as a method to avoid overfitting. Overfitting noisy data in the data set of Exercise session 1 occurs when the resulted fit shows large gradients in order to reproduce every single data points (which minimizes the cost function). However, the principle of Occam's razor dictates that overcomplex systems should be avoided for simple problems. When complexity rises, errors on the test set might increase, indicating overfitting.

1 'Pure' Bayesian training versus 'trainbr'

In the `perbayes` script, the weight values correspond the the MAP, i.e. the maximum of the posterior

$$\log P(\vec{w}|\mathcal{D}) = \log P(\mathcal{D}|\vec{w}) + \log P(\vec{w}) - \log P(\mathcal{D})\,. \tag{1}$$

for the data $\mathcal{D}$. Since the last term does not depend on the weight vector $\vec{w}$, we do not consider it in the following. Note that, if one compares this with the cost function M in the assignment, we note that no hyper-parameters α and β are included here. Therefore, we refer to this as the 'pure' Bayesian algorithm. This is a restriction compared to the more general `trainbr` algorithm discussed below.

In order to understand the need for the hyper parameters, we vary the number of data points in the data set. For a high number of data points, the influence of the prior on the posterior is negligible. However, the likelihood calculates the probability of generating a set of discrete values (0 and 1), from a continuous distribution (a sigmoid in this case). Naturally, the optimization converges to a sigmoid function which is as sharp as possible, corresponding to a weight vector in the outskirts of its domain. In the limit of an infinite data set, the modulus of the weight vector is over estimated, and the direction is incorrect. The latter is due to the fact that the modulus of the weight vector is largest when $|\vec{w}| = \sqrt{2}$. Hence, a sharp boundary is favoured over a correctly aligned boundary. In the case at hand, overfitting of the network corresponds to an overestimated $|\vec{w}|$. This effect we wish to counteract by introduction of the prior, wish gives a higher prior probability of obtaining smaller values of $|\vec{w}|$.

For similar reasons, the MAP does not always result in a weight vector which is close to the weight vector of the original network. In most cases, the weight vector has an over estimated modulus[1]. An example is shown in Fig 1. Adjusting the effect of the prior properly can reduce this effect. Hence, introducing hyper parameters to scale the effect of the data and prior can improve the fitting. This is the case for the Bayesian regularization algorithm `trainbr`.

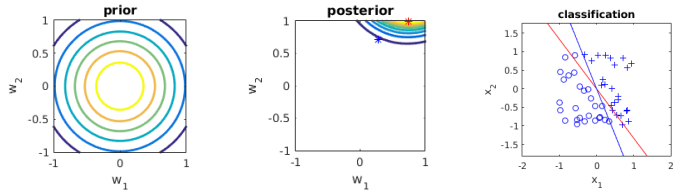

Figure 1: Prior, posterior and classification with corresponding separation line of the original (blue) and MAP network (red). Some (but not many) samples are misclassified.

2 Bayesian regularization using 'trainbr'

We now focus on the data set used in the previous report to discuss overfitting, and how Bayesian regulation my help reduce it. An example of the effect of a Bayesian regularization term is illustrated in Fig. 2 where we compare the Maximum-a-posteriori result with a Maximum-Likelihood result. In the case at hand, in essence, we wish to approximate the underlying function $\sin(x)$ of the data, rather than the data set itself. In the Bayesian framework, regularization can be implemented in a natural way by opting for a parameter prior which has the bulk of its probability close to small values of the network weights.

[1] For these reasons, angles of 45, 135 and 225 and 315 degrees are favoured. However, this effect is smaller for a lower number of data points.

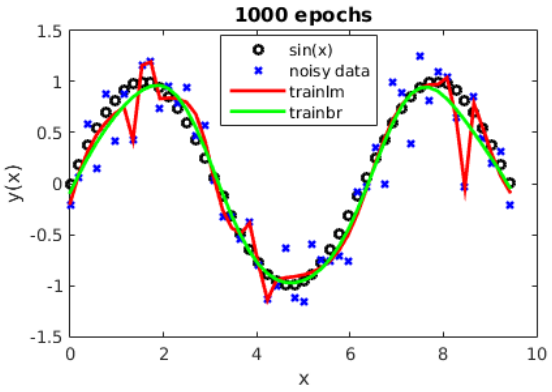

Figure 2: Illustration of an overfitted network trained with the Levenberg-Marquadt algorithm (red), compared to the results of the Bayesian learning (green) for a data set with $h(x) = \sin(x) + \mathcal{N}(\sigma)$ where $\sigma = 0.3$. Both networks have the same architecture, which includes 10 neurons in the hidden layer.

In the last report we characterized generalization and overfitting via the network's variance[2]. We can therefore use the figure used in the previous report, now shown in Fig. 3, and compare the `trainbr` algorithm with the other training algorithms. Figure 3 compares the bias and variance as a function of the complexity of the network for different learning algorithms.

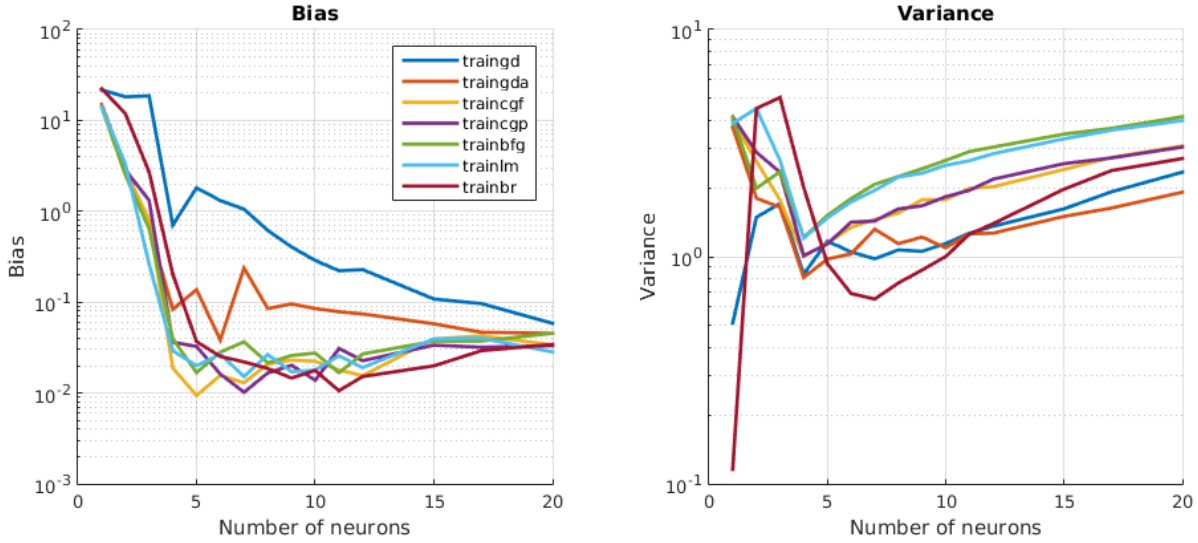

Figure 3: Bias and variance plot as a function of the complexity of the neural network.

An overfitted network corresponds to a low bias but high variance in Fig. 3. Assuming convergence has been reached, a large variance indicates that the resulting network depends largely on the data set to which it is fitted, and hence, to the noise it includes. The effect of the Bayesian regulation term is most clear in the interval of medium complexity (5-10 hidden neurons). Both the bias and variance remain low, indicating that the `trainbr` algorithm allows for a more accurate function reconstruction, while avoiding noise fitting (see again Fig. 2 for an illustration). Interestingly enough however, Fig. 3 also clearly shows the limited range of application of the Bayesian regulation. When the complexity is high enough, the regulation term has a negligible effect, and also `trainbr` will start overfitting.

[2]Naturally, we could have also done this using a test set.