

Instruction for Computer Lab Experiments: Generalization and model order selection

©Signals and Systems Group, Uppsala University

October, 2004

Introduction

In this course we have so far encountered a number of situations in which we tried to find suitable values for some parameters, \mathbf{w} , so that a function, $g(\mathbf{x}, \mathbf{w})$, defined by these parameters will match well to a given training data set, $\mathbf{T} = [t_1, \dots, t_N]^T$. Let us assume that our training data have been formed according to

$$t_k = h(\mathbf{x}_k) + e_k, \quad (1)$$

where e_k is a zero mean Gaussian noise term with variance σ^2 and $h(\mathbf{x}_k)$ is some unknown function. We try to approximate $h(\mathbf{x})$ using a model function $g(\mathbf{x}, \mathbf{w})$.

In this lab, we have chosen the underlying true function to be a polynomial of degree four,

$$h(\mathbf{x}) = -0.6x^2 + 0.1x^4, \quad (2)$$

and the model $g(\mathbf{x}, \mathbf{w})$ is either a polynomial or a two layer MLP with linear output neuron but we could equally well have chosen a radial basis function (RBF) net or any model typically used for function approximation.

Common to all these modeling problems is that we need not only to determine the parameters, we also need to find a suitable model structure (or model order). For the polynomial we need to decide which polynomial degree to use. For the MLP we generally need to decide how many layers the network should have (although in this lab it fixed to be two), and how many neurons each layer should consist of. For an RBF-net we would need to decide the number of basis functions.

The importance of treating this problem in a good way becomes obvious if we understand the concepts of generalization, bias, and variance. We should keep in mind that our main interest is typically in finding a good approximation of the function $h(\mathbf{x})$. Unless the model $g(\mathbf{x}, \mathbf{w})$ can be physically motivated¹, we are not really interested in the parameter values themselves. We are just interested in the output function values, and the model parameters are merely carriers of information between our training data set and the function values for previously unseen \mathbf{x} .

The problem of drawing conclusions about unseen data based on training data is called generalization and in this computer exercise you can study how the generalization properties depend on the number of data points, N , the noise level in the training data, σ^2 , and the model order, m . You will probably realize that the problem of model order selection can many times be quite important and that there is a need for methods that either help us to find, in some sense, the best model order or to help us make the model order choice less critical.

¹then with some parameters that can be interpreted as something more than just numbers

The computer exercise is organized as follows: In Section 1, the generalization problem is illustrated in a number of ways, showing explicitly how the individual terms *Bias*² and *Variance* depend on the model order and showing the effects of under- and overparameterization. In Section 2, we examine how we can use validation to find a suitable model order and, finally, in Section 3, regularization (interpreted as MAP-estimation) is illustrated.

For each section there are a few scripts that you can run. Try to describe as many aspects of the results as possible and write down your conclusions. *Note that you can and should modify the parameters of interest in the code.* The laboratory assistant will guide you in the right directions if you run out of ideas of what to look for.

Some results will hopefully be in complete agreement with your intuition whereas some of the results might be, at least at a first glance, somewhat unintuitive. The goal with the exercise is to confirm the good intuition that you might already have and to refine your intuition in cases where it seems to be needed.

1 The generalization problem.

All training algorithms in this section aim at minimizing the usual sum squared errors.

- **Intro:** A simple startup script, introducing you to the concepts. Here you see how polynomials with degrees between $m = 1$ and $m = 6$ are fitted using least squares to three data sets. For each fit you also see: (1) The training error, that is the error which was explicitly minimized (2) the average squared error between the model and the true function $h(\mathbf{x})$ (based on 1000 previously unseen examples) and (3) the average squared error between the model and $t(\mathbf{x}) = h(\mathbf{x}) + e$. In **Intro**, the points on the x -axis are fixed and equally spaced. (cf. an experimental design where you can control the inputs).
- **PolXfix:** Much the same as **Intro** but now we calculate (by means of Monte-Carlo simulations) the squared errors that were shown in **Intro** *averaged* over several training data sets.
- **PolXrandom:** Almost the same as **PolXfix** but now also the x -values are picked at random.
- **MLPXrandom:** Much the same as **PolXrandom** but $g(\cdot)$ is implemented using an MLP instead of a polynomial. This is a relatively slow script (the time depends on the number of Monte-Carlo runs, N_{MC} that you find in the code).

2 Validation

To save time, only the polynomial models are considered here. Again, the parameters are obtained through least squares fitting.

- **PolXrandomHO:** Illustrating the “hold-out method”.
- **PolXrandomCV:** Illustrating the technique of cross-validation, which has as a special case “leave-one-out”.

3 Regularization. (Interpreted as MAP-estimation of the parameters)

Training performed in these scripts do not explicitly aim at minimizing the sum squared error. An extra regularization term is included in the training criteria, punishing large parameter values. The regularization can be interpreted as if we are performing MAP estimation instead of ML estimation² and the extra term is then the negative log of a pdf describing our prior information on the parameters. For the polynomial we here assume that (i) the parameters are independent (ii) zero mean Gaussian with individual standard deviations. Altogether, this leads to a regularization term that is weighted sum of squared coefficients. For the MLP, the regularization term is $\alpha \sum_i w_i^2$, where the sum is over all weights and biases in the MLP. Unfortunately, we have limited freedom in choosing the regularization parameter α since Matlab's Neural Network Toolbox automatically finds a suitable value for us using a certain rule of thumb.

- `IntroReg`: Same as `Intro` but now we can regularize the parameters by performing MAP estimation.
- `PolXrandomReg` Same as `PolXrandom` but with regularization.
- `MLPXrandomReg` Same as `MLPXrandom` but with regularization (rel. slow script).

Good luck!

²as we could interpret the earlier scripts if e_n is Gaussian.