The Problem of Overfitting with Maximum Likelihood

In the previous example, continuing training to find the absolute maximum of the likelihood produced overfitted results. The effect is much bigger if there are more inputs and/or more hidden units. Here are two ways of fixing this:

Add a penalty to the log likelihood. Similar to other penalty methods we have seen before. For an MLP with one hidden layer, a suitable penalty might be

$$-\lambda_1 \sum_{k=1}^{q} \beta_k^2 \;-\; \lambda_2 \sum_{k=1}^{q} \sum_{j=1}^{p} \gamma_{k,j}^2$$

We have to set two constants controlling the penalty, λ_1 and λ_2, perhaps using cross validation. (Setting $\lambda_1 = \lambda_2$ may not be reasonable.)

Stop doing gradient descent before a maximum is reached. We saw in the example that gradient descent spent a long time in a reasonable solution (not overfitted) before things went a bit crazy. Maybe we can stop at this point. Some good results reported using an MLP depend on having done this without meaning to (they just stopped when they lost patience with gradient descent).

Early Stopping Using Cross Validation

We can tell when to stop gradient descent optimization using a set of validation cases that's separate from the cases used to compute the gradient.

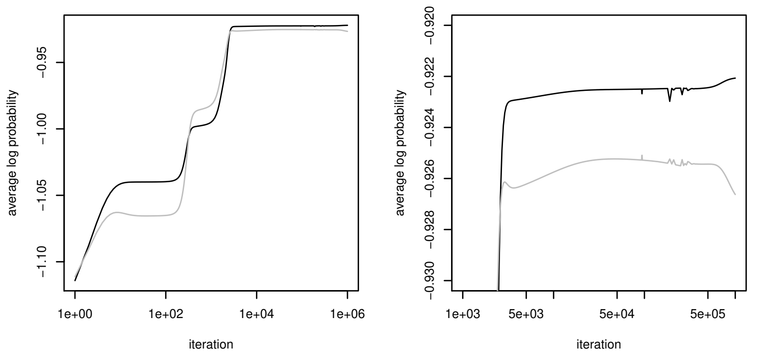
Here's the procedure:

1. Randomly divide the training set into an estimation set and a validation set — eg, $0.8n$ cases in the estimation set and $0.2n$ cases in the validation set.
2. Randomly initialize the parameters to values near zero.
3. Repeatedly do the following:
   - Compute the gradient of the log likelihood using the estimation set.
   - Update the parameters by adding $\eta\nabla\ell$.
   - Compute the log probability of y given x for the validation cases.

   Stop when the average log probability for validation cases is substantially less than the maximum of the values found previously (definitely getting worse).
4. Make predictions for test cases using the parameter values from the loop above that gave the highest average log probability to the validation cases.

An Example of Early Stopping Using Cross Validation

I tried same example as before, but with 75 of the 100 training cases used for estimation (computing the gradient) and 25 used for validation (deciding when to stop / which parameters to use). I used 20 hidden units this time, and set $\eta = 0.2$.

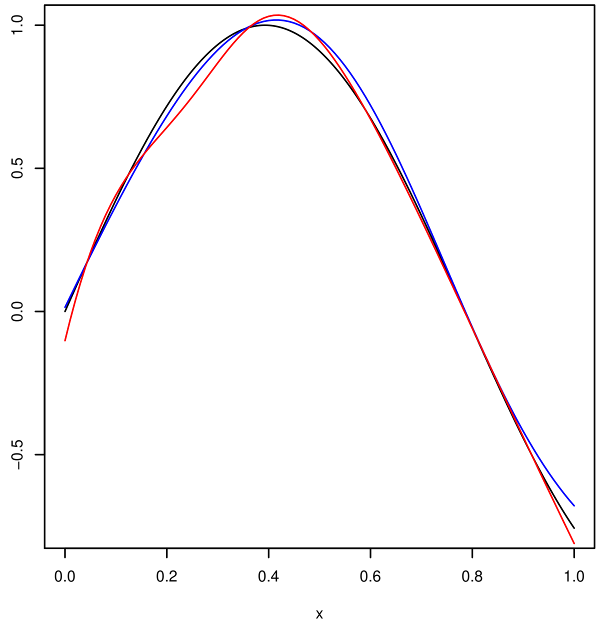


I divide total log probability by the number of cases in these plots, so that 75 vs. 25 won't matter. The average for estimation cases is shown in black, for validation cases in grey. The highest log probability for validation cases is at iteration 94000.

The Function Computed by the Network Selected

Here is the true regression function, $\sin(4x)$, in black, the regression function defined by the network at iteration 94000, selected by cross validation, and the last network, at iteration 1000000.

Cross validation worked this time, but 25 validation cases is really to few for reliable results.



Advantages of Early Stopping

Early stopping using a validation set has some advantages over a penalty method, in which you set λ_1 and λ_2 using N-fold cross validation, then train again on all the data with the λ_1 and λ_2 you choose.

With early stopping:

- You only have to train the network *once* — not once for each setting of λ_1 and λ_2 you consider, and then again on all the training data to get the final parameters. Important if training takes a week!
- The performance measure from cross-validation applies directly to the actual set of network parameters you will use, not to values of λ_1 and λ_2 that may not do the same thing for a different local maximum.

Disadvantages of Early Stopping

Early stopping also has some problems:

- It's very *ad hoc* — what's the justification for this procedure?
- It depends on details of the optimization method — eg, results could be different with a different η.
- In particular, we might want to use a different η for βs than for γs — sort of like using different λs for a penalty method.
- Some of the training data is used only to decide when to stop — this seems wasteful.

We could try to solve the third problem by doing training runs using different values of η for the βs and for the γs, and then pick the best parameters (using the validation set) from all these runs.

The fourth problem can also be solved by doing more training runs...

Using an Ensemble of Early-Stopped Networks

Rather than train just once, with a single split into estimation and validation sets, we can train several times, with several splits, to create an *ensemble* of networks.

It seems best to split the training set into N equal portions, and then train N times. Each training run uses $N-1$ portions of the training set for estimation, and one portion for validation (to choose which parameters from the run to use).

We get N sets of parameters this way. To make predictions for test cases, we *average* the predictions from all N networks. (We average network outputs for regression problems, and average class probabilities for classification problems.)

This way we use the whole training set for estimation. Averaging several network outputs also reduces the effect of any single peculiar training run.