Improved training methods: 02457 Non-Linear Signal Processing, Exercise 6

This exercise is based om C. M. Bishop: *Neural Networks for Pattern Recognition* chapter 7. The objective of this exercise is to become familiar with optimization methods for nonlinear neural network models. The exercise will focus on the pseudo(-Gauss)-Newton method and the conjugate gradient algorithms.

Print and comment on the figures produced by the software as outlined below at the **Checkpoints**.

You are going to use the “Neural Regression” Matlab toolbox programmed at Digital Signal Processing section IMM DTU. The main Matlab function for training a two-layer feed-forward neural network with conjugate gradient and pseudo-Gauss-Newton is called `nr_trainx`.

Optimization procedures

There is a host of algorithms for non-linear system optimization. Unfortunately this reflects the application specific nature of the problem, no algorithm is uniformly superior. A subset of important algorithms is shown in table 1.

1st	2nd	Name
–	–	Amoebe / simplex / Nelder-Mead
+	–	**Gradient descent**
+	–	Gradient descent with momentum
+	–	Natural gradient
+	–	Conjugate gradient algorithm
		— **Hestenes-Stiefel**
		— **Fletcher-Reeves**
		— **Polak-Ribiere**
		— Scaled conjugate gradient
+	–	Quasi-Newton
		— Davidson-Fletcher-Powell (DFP)
		— Rank-one-formula
		— Broyden-Fletcher-Goldfarb-Shanno (BFGS)
+	(+)	**Pseudo(-Gauss)-Newton**
+	(+)	Gauss-Newton
+	+	Levenberg-Marquardt
+	+	Newton(-Ralphson)

Table 1: Some optimization algorithms.

Most of the algorithms in table 1 use an iterative scheme where the parameters $\mathbf{w}$ are initialized to some values and then a step is taken to a new place

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{1}$$

The usual *gradient descent* (from the last exercise) is the negative gradient multiplied with a suitable learning rate η:

$$\Delta \mathbf{w} = -\eta \, \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \tag{2}$$

One can develop the second order algorithms (Newton, Levenberg-Marquardt, Gauss-Newton, pseudo-Gauss-Newton) from a Taylor expansion up the second order term of the costfunction E around $\hat{\mathbf{w}}$:

$$E(\mathbf{w}) = E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})'\mathbf{g}_{\hat{\mathbf{w}}} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})'\mathbf{H}_{\hat{\mathbf{w}}}(\mathbf{w} - \hat{\mathbf{w}}) + \ldots \tag{3}$$

$\mathbf{g}_{\hat{\mathbf{w}}}$ is the first order derivative / gradient of the cost function in $\hat{\mathbf{w}}$ and $\mathbf{H}_{\hat{\mathbf{w}}}$ is the second order derivative — the Hessian — of the costfunction in $\hat{\mathbf{w}}$. The first order derivative of in $\mathbf{w}$ is:

$$\nabla E(\mathbf{w}) = \mathbf{g}_{\hat{\mathbf{w}}} + \mathbf{H}_{\hat{\mathbf{w}}}(\mathbf{w} - \hat{\mathbf{w}}) \tag{4}$$

We want to find a local minimum $\mathbf{w} = \mathbf{w}_0$. The gradient should be zero there: $\nabla E(\mathbf{w}_0) = 0$, which means that we now can isolate $\mathbf{w}_0$:

$$\mathbf{w}_0 = \hat{\mathbf{w}} + (-\mathbf{H}_{\hat{\mathbf{w}}}^{-1}\mathbf{g}_{\hat{\mathbf{w}}}) \tag{5}$$

Taking this step is the (full) Newton algorithm. The other second order methods (and the quasi-Newton methods) use some kind of approximation to the Hessian, e.g., the *pseudo-Gauss-Newton* uses only the diagonal of the Hessian (here for each variable w_i in $\mathbf{w}$):

$$\Delta w_i = -\frac{\partial E}{\partial w_i} \bigg/ \frac{\partial^2 E}{\partial w_i^2} \tag{6}$$

Conjugate gradient algorithms construct a series of *conjugate* directions $\mathbf{d}$. These are direction that satisfy the following condition:

$$\mathbf{d}'_{j+1}\mathbf{H}\mathbf{d}_j = 0 \tag{7}$$

There are three classic conjugate gradient algorithms, Hestenes-Stiefel, Fletcher-Reeves and Polak-Ribiere, which construct the series of conjugate directions using the following equations ($\mathbf{g}$ is the gradient):

$$\begin{aligned}
\mathbf{d}_{j+1} &= -\mathbf{g}_{j+1} + \frac{\mathbf{g}'_{j+1}(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{d}_j{}'(\mathbf{g}_{j+1} - \mathbf{g}_j)}\mathbf{d}_j && (8)\\
\mathbf{d}_{j+1} &= -\mathbf{g}_{j+1} + \frac{\mathbf{g}'_{j+1}\mathbf{g}_{j+1}}{\mathbf{g}'_j\mathbf{g}_j}\mathbf{d}_j && (9)\\
\mathbf{d}_{j+1} &= -\mathbf{g}_{j+1} + \frac{\mathbf{g}'_{j+1}(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j{}'\mathbf{g}_j}\mathbf{d}_j && (10)
\end{aligned}$$

The conjugate gradient algorithms usually require that the costfunction is minimized along the direction, thus a *line search* is performed.

Optimization in the neural regression toolbox

The neural regression toolbox implements five different optimization algorithms presently: Gradient descent, pseudo-Gauss-Newton and three conjugate gradient algorithms: Hestenes-Stiefel (HS), Fletcher-Reeves (FR) and Polak-Ribiere (PR). The function `nr_trainx` implements them all and `nr_train` only implements the two first. The gradient descent algorithm chooses the (negated) gradient as its direction and the step size (Bishop: learning rate η) is determined by line search using a sort of bisection: The gradient decent starts with a step that is as long a gradient (i.e., $\eta = 1$) and then halves it until if the costfunction is decreasing. The pseudo-Gauss-Newton starts with a length determined by equation 6. The line search for both of these algorithms is implemented in `nr_linesear`. The pseudo-Gauss-Newton presently starts with 10 gradient descent step so that it (hopefully!) will get into a region where the second order derivative is well-behaved.

The conjugate gradient algorithms use another type of line search algorithm: quadratic (parabolic) and cubic interpolation and extrapolation (Bishop, pp 273-274). The line search is inexact and the stop criterion for it is the so-called Wolfe-Powell condition.

This kind of line search is imlemented with `nr_linesearch` (not the same as `nr_linesear`!).

The neural network is run on the sunspot data. There are 12 inputs plus a bias unit, 3 hidden units and one output.

Checkpoint 6.1:

Use the function `main6a.m` to plot the surface of the costfunction as a function of two of the largest weights. The plot also contains three trace from optimization: Two with gradient descent (without line search) distinguished by a small and a large step size and one pseudo-Gauss-Newton. Which one is which?

Checkpoint 6.2:

Make a flow chart and describe the calculations and functions in the neural net program `main6b.m`. Plot the evolution of the training error and the gradient norms for the gradient descent using `main6b.m`, for 3 and 8 hidden units. Describe the evolution of the gradient norm curve: Are there relations between "events" in the training error curve and in the gradient norm curve?

Checkpoint 6.3:

Use the program `main6c.m` to investigate the speed of convergence for the different algorithms. The program runs the algorithms a couple of times (each with a different seed for the initialization of the weights) and computes the average of the costfunction value. The evolution of the costfunctions is available in the variable `E` Determine which is the best algorithm and comment on the shape of curves.

DTU, 1999, 2003, Finn Årup Nielsen and Lars Kai Hansen.