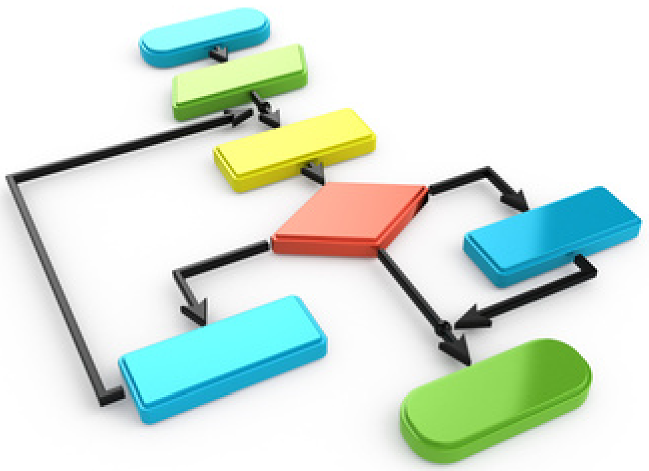
5 algorithms to train a neural network

By Alberto Quesada, Artelnics.

The

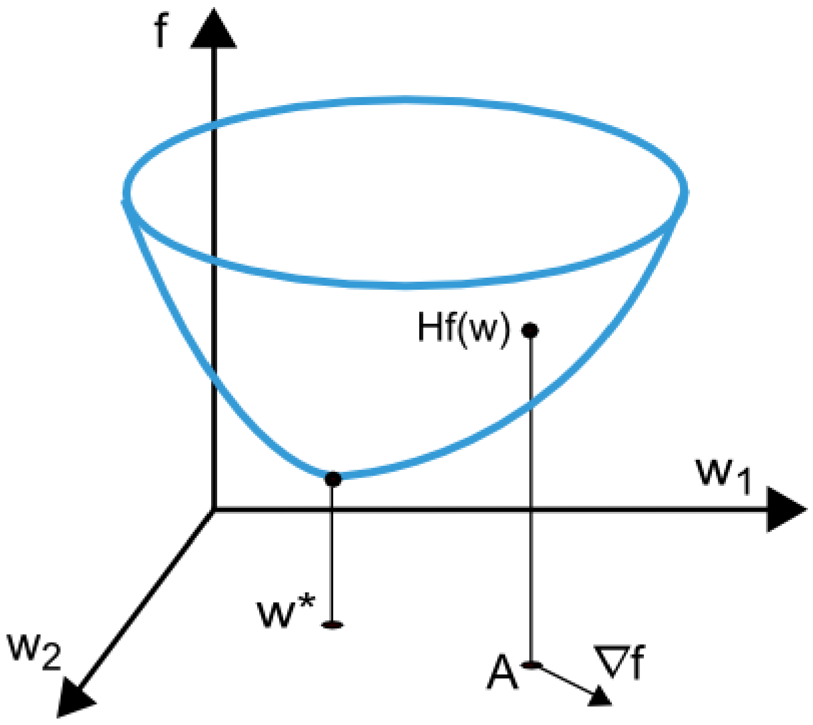


procedure used to carry out the learning process in a neural network is called the training algorithm. There are many different training algorithms, whith different characteristics and performance.

Problem formulation

The learning problem in neural networks is formulated in terms of the minimization of a loss function, f. This function is in general, composed of an error and a regularization terms. The error term evaluates how a neural network fits the data set. On complexity of the neural network.

The loss function depends on the adaptative parameters (biases and synaptic weights) in the neural network. We can conveniently group them together into a single n-dimensional weight vector w. The picture below represents the loss function f(w).



As we can see in the previous picture, the point w* is minima of the loss function. At any point A, we can calculate the first and second derivatives of the loss function. The first derivatives are gropued in the gradient vector, whose elements can be written

$$\boxed{\nabla_i f(w) = df/dw_i \ (i = 1,\ldots,n)}$$

can be grouped in the Hessian matrix,

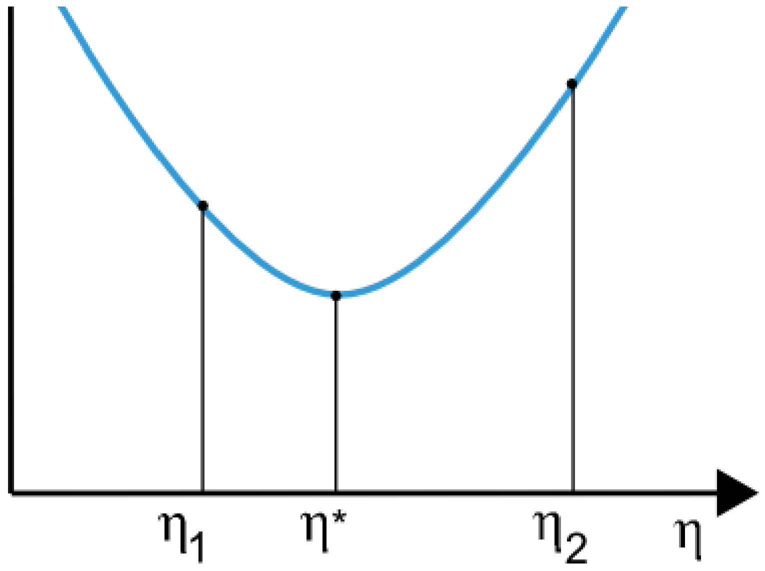
$$\boxed{H_{i,j}f(w) = d^2f/dw_i \cdot dw_j \ (i,j = 1,\ldots,n)}$$

The problem of minimizing continuous and differentiable functions of many variables has been widely studied. Many of the conventional approaches to this problem are directly applicable to that of training neural networks.

One-dimensional optimization

Although the loss function depends on many parameters, one-dimensional optimization methods are of great importance here. Indeed, they are are very often used in the training process of a neural network.

Indeed, many training algorithms first compute a training direction d and then a traning rate η that minimizes the loss in that direction, f(η). The next picture illustrates this one-dimensional function.



The points η_1 and η_2 define an interval that contains the minimum of f, η^*.

In this regard, one-dimensional optimization methods search for the minimum of a given one-dimensional function. Some of the algorithms which are widely used are the golden section method and the Brent's method. Both reduce the bracket of a minumum until the distance between the two outer points in the bracket is less than a defined tolerance.

Multidimensional optimization

The learning problem for neural networks is formulated as searching of a parameter vector w^* at which the loss function f takes a minimum value. The necessary condition states that if the neural

The loss function is, in general, a non linear function of the parameters. As a consequence, it is not possible to find closed training algorithms for the minima. Instead, we consider a search through the parameter space consisting of a succession of steps. At each step, the loss will decrease by adjusting the neural network parameters.

In this way, to train a neural network we start with some parameter vector (often chosen at random). Then, we generate a sequence of parameters, so that the loss function is reduced at each iteration of the algorithm. The change of loss between two steps is called the loss decrement. The training algorithm stops when a specified condition, or stopping criterion, is satisfied.

Now, we are going to describe the most importat training algorithms for neural networks.

Gradient descent | Newton's method | Conjugate gradient | Quasi Newton | Levenberg Marquardt

1. Gradient descent

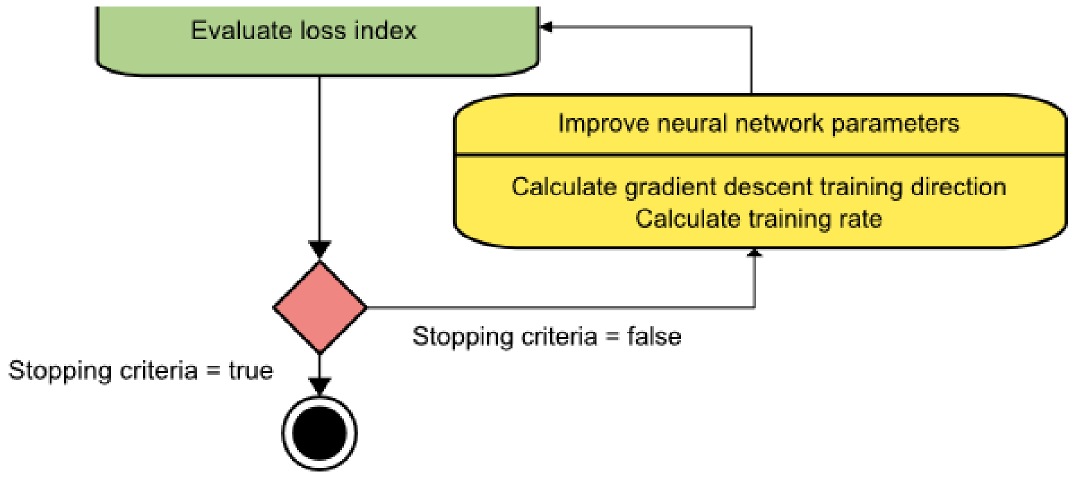
Gradient descent, also known as steepest descent, is the simplest training algorithm. It requires

Let denote $f(w_i) = f_i$ and $\nabla f(w_i) = g_i$. The method begins at a point w_0 and, until a stopping criterion is satisfied, moves from w_i to w_{i+1} in the training direction $d_i = -g_i$. Therefore, the gradient descent method iterates in the following way:

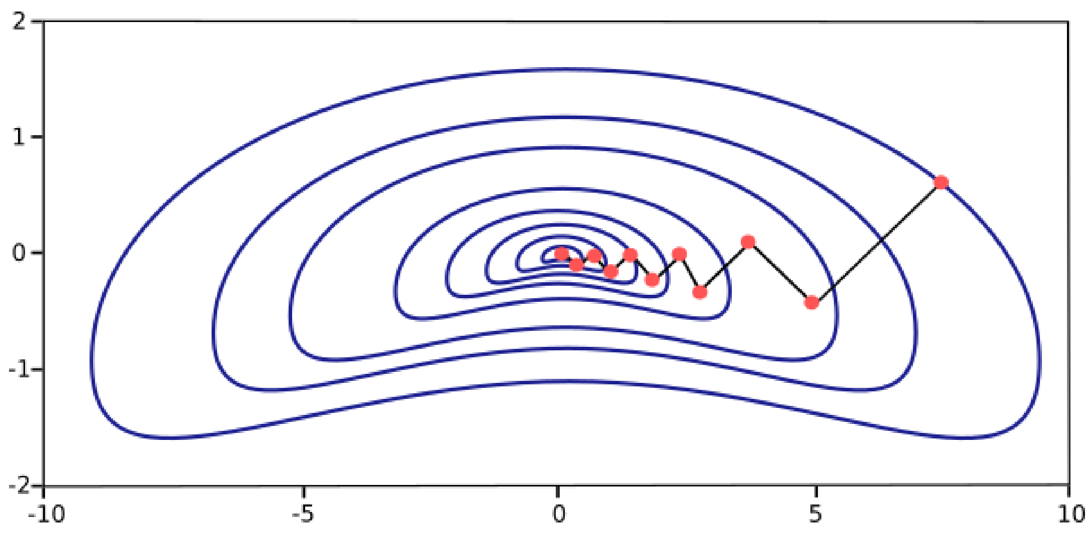
$$\boxed{w_{i+1} = w_i - d_i \cdot \eta_i, \quad i=0,1,\ldots}$$

The parameter η is the training rate. This value can either set to a fixed value or found by one-dimensional optimization along the training direction at each step. An optimal value for the training rate obtained by line minimization at each successive step is generally preferable. However, there are still many software tools that only use a fixed value for the training rate.

The next picture is an activity diagram of the training process with gradient descent. As we can see, the parameter vector is improved in two steps: First, the gradient descent training direction is computed. Second, a suitable training rate is found.


Evaluate loss index
Improve neural network parameters
Calculate gradient descent training direction
Calculate training rate
Stopping criteria = false
Stopping criteria = true

The gradient descent training algorithm has the severe drawback of requiring many iterations for functions which have long, narrow valley structures. Indeed, the downhill gradient is the direction in which the loss function decreases most rapidly, but this does not necessarily produce the fastest convergence. The following picture illustrates this issue.



Gradient descent is the recommended algorithm when we have very big neural networks, with many it does not store the Hessian matrix (size n^2).

2. Newton's method

The Newton's method is a second order algorithm because it makes use of the Hessian matrix. The objective of this method is to find better training directions by using the second derivatives of the loss function.

Let denote $f(w_i) = f_i$, $\nabla f(w_i) = g_i$ and $Hf(w_i) = H_i$. Consider the quadratic approximation of f at w_0 using the Taylor's series expansion

$$\boxed{f = f_0 + g_0 \cdot (w - w_0) + 0.5 \cdot (w - w_0)^2 \cdot H_0}$$

H_0 is the Hessian matrix of f evaluated at the point w_0. By setting g equal to 0 for the minimum of f(w), we obtain the next equation

$$\boxed{g = g_0 + H_0 \cdot (w - w_0) = 0}$$

Therefore, starting from a parameter vector w_0, Newton's method iterates as follows

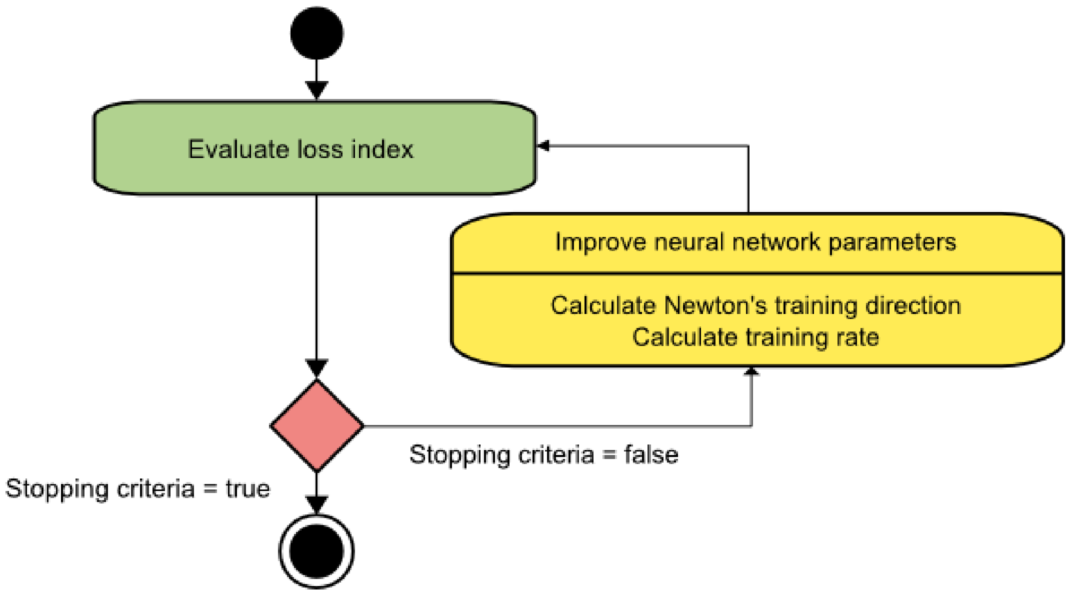
$$\boxed{w_{i+1} = w_i - H_i^{-1} \cdot g_i, \quad i=0,1,...}$$

towards a maximum rather than a minimum. This occurs if the Hessian matrix is not positive definite. Thus, the function evaluation is not guaranteed to be reduced at each iteration. In order to prevent such troubles, the Newton's method equation usually modified as:

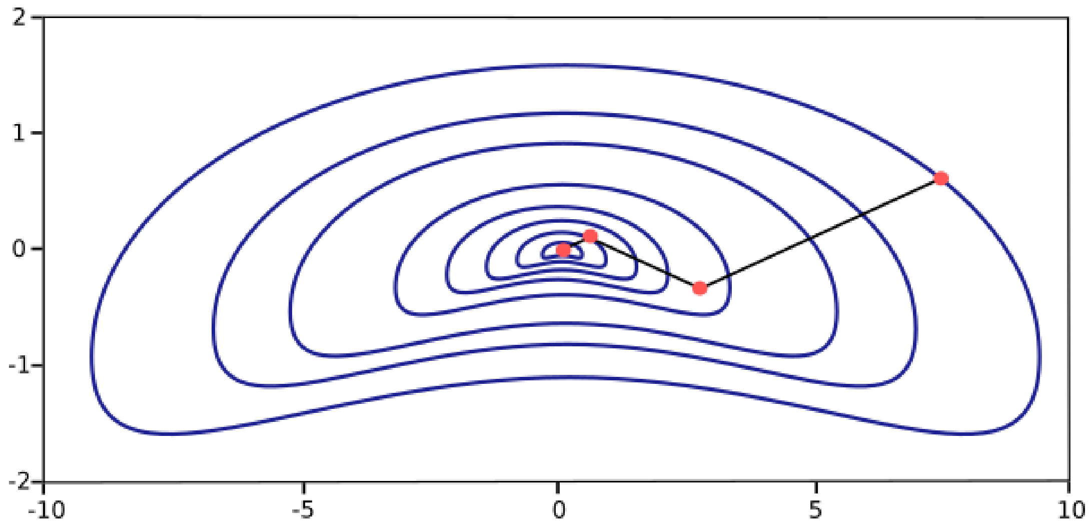
$$\boxed{w_{i+1} = w_i - (H_i^{-1} \cdot g_i) \cdot \eta_i, \quad i=0,1,\ldots}$$

The training rate, η, can either set to a fixed value or found by line minimization. The vector $d = H_i^{-1} \cdot g_i$ is now called the Newton's training direction.

The state diagram for the training process with the Newton's method is depicted in the next figure. Here improvement of the parameters is performed by obtaining first the Newton's training direction and then a suitable training rate.


Evaluate loss index
Improve neural network parameters
Calculate Newton's training direction
Calculate training rate
Stopping criteria = false
Stopping criteria = true

requires less steps than gradient descent to find the minimum value of the loss function.



However, the Newton's method has the difficulty that the exact evaluation of the Hessian and its inverse are quite expensive in computational terms.

3. Conjugate gradient

The conjugate gradient method can be regarded as something intermediate between gradient descent and Newton's method. It is motivated by the desire to accelerate the typically slow convergence associated with gradient descent. This method also avoids the information requirements associated with the evaluation, storage, and inversion of the Hessian matrix, as required by the Newton's method.

which produces generally faster convergence than gradient descent directions. These training directions are conjugated with respect to the Hessian matrix.

Let denote d the training direction vector. Then, starting with an initial parameter vector w_0 and an initial training direction vector $d_0 = -g_0$, the conjugate gradient method constructs a sequence of training directions as:

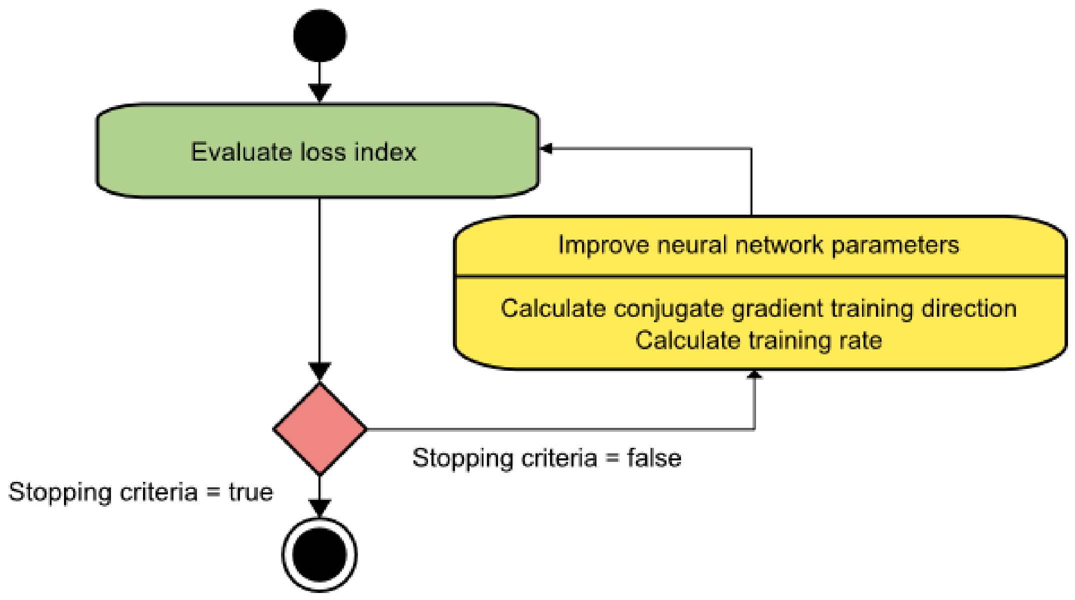
$$\boxed{d_{i+1} = g_{i+1} + d_i \cdot \gamma_i, \quad i=0,1,\ldots}$$

Here γ is called the conjugate parameter, and there are different ways to calculate it. Two of the most used are due to Fletcher and Reeves and to Polak and Ribiere. For all conjugate gradient algorithms, the training direction is periodically reset to the negative of the gradient.

The parameters are then improved according to the next expression. The training rate, η, is usually found by line minimization.

$$\boxed{w_{i+1} = w_i + d_i \cdot \eta_i, \quad i=0,1,\ldots}$$

The picture below depicts an activity diagram for the training process with the conjugate gradient. Here and then suitable training rate in that direction.


Evaluate loss index
Improve neural network parameters
Calculate conjugate gradient training direction
Calculate training rate
Stopping criteria = false
Stopping criteria = true

This method has proved to be more effective than gradient descent in training neural networks. Since it does not require the Hessian matrix, conjugate gradient is also recommended when we have very big neural networks.

4. Quasi-Newton method

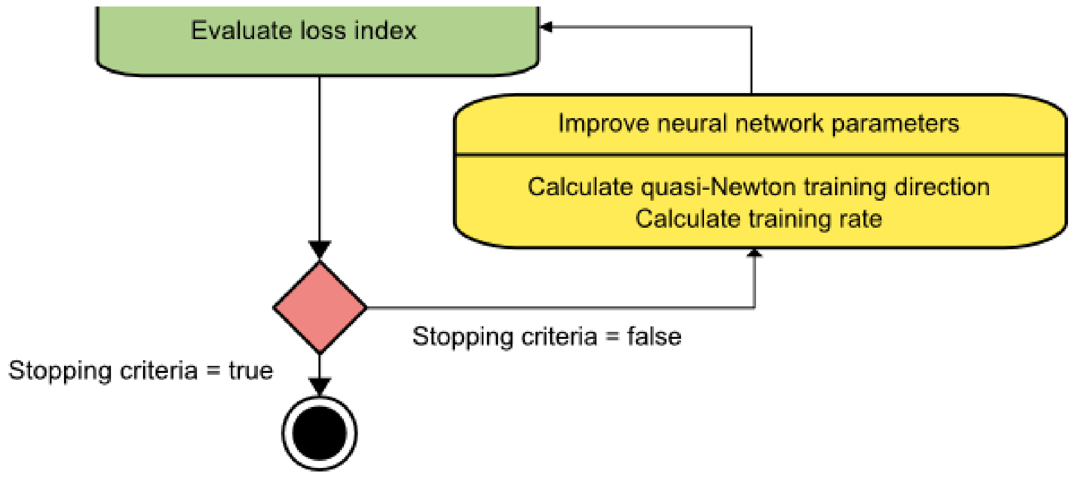
Application of the Newton's method is computationally expensive, since it requires many operations to evaluate the Hessian matrix and compute its inverse. Alternative approaches, known as quasi-Newton or variable metrix methods, are developed to solve that drawback. These methods, instead of calculating the Hessian directly and then evaluating its inverse, build up an approximation to only information on the first derivatives of the loss function.

The Hessian matrix is composed of the second partial derivatives of the loss function. The main idea behind the quasi-Newton method is to approximate the inverse Hessian by another matrix G, using only the first partial derivatives of the loss function. Then, the quasi-Newton formula can be expressed as:

$$\boxed{w_{i+1} = w_i - (G_i \cdot g_i) \cdot \eta_i, \quad i=0,1,\ldots}$$

The training rate η can either set to a fixed value or found by line minimization. The inverse Hessian approximation G has different flavours. Two of the most used are the Davidon–Fletcher–Powell formula (DFP) and the Broyden–Fletcher–Goldfarb–Shanno formula (BFGS).

The activity diagram of the quasi-Newton training process is illustrated bellow. Improvement of the parameters is performed by first obtaining the quasi-Newton training direction and then finding a satisfactory training rate.


Evaluate loss index
Improve neural network parameters
Calculate quasi-Newton training direction
Calculate training rate
Stopping criteria = false
Stopping criteria = true

This is the default method to use in most cases: It is faster than gradient descent and conjugate gradient, and the exact Hessian does not need to be computed and inverted.

5. Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm, also known as the damped least-squares method, has been designed to work specifically with loss functions which take the form of a sum of squared errors. It works without computing the exact Hessian matrix. Instead, it works with the gradient vector and the Jacobian matrix.

Consider a loss function which can be expressed as a sum of squared errors of the form

$$\boxed{f = \sum e_i^2, \quad i=0,\ldots,m}$$

We can define the Jacobian matrix of the loss function as that containing the derivatives of the errors with respect to the parameters,

$$\boxed{J_{i,j}f(w) = de_i/dw_j \ (i = 1,\ldots,m \ \& \ j = 1,\ldots,n)}$$

Where m is the number of instances in the data set and n is the number of parameters in the neural network. Note that the size of the Jacobian matrix is m·n.

The gradient vector of the loss function can be computed as:

$$\boxed{\nabla f = 2\ J^T \cdot e}$$

Here e is the vector of all error terms.

Finally, we can approximate the Hessian matrix with the following expression.

$$\boxed{Hf \approx 2\ J^T \cdot J + \lambda I}$$

Where λ is a damping factor that ensures the positiveness of the Hessian and I is the identity matrix.

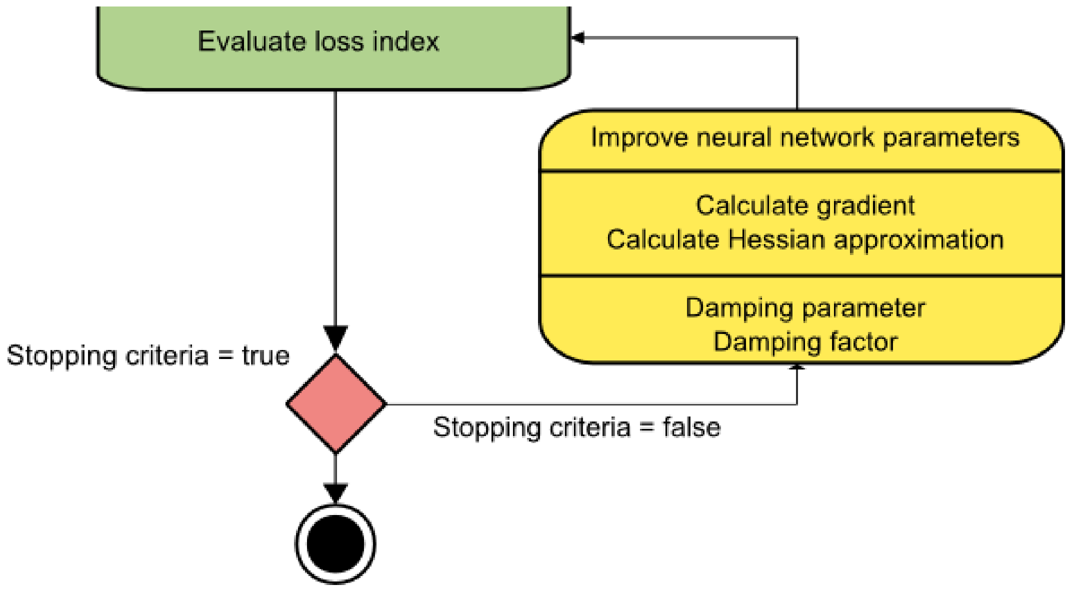
algorithm

$$w_{i+1} = w_i - (J_i^T \cdot J_i + \lambda_i I)^{-1} \cdot (2\, J_i^T \cdot e_i), \quad i=0,1,\ldots$$

When the damping parameter λ is zero, this is just Newton's method, using the approximate Hessian matrix. On the other hand, when λ is large, this becomes gradient descent with a small training rate.

The parameter λ is initialized to be large so that first updates are small steps in the gradient descent direction. If any iteration happens to result in a failure, then λ is increased by some factor. Otherwise, as the loss decreases, λ is decreased, so that the Levenberg-Marquardt algorithm approaches the Newton method. This process typically accelerates the convergence to the minimum.

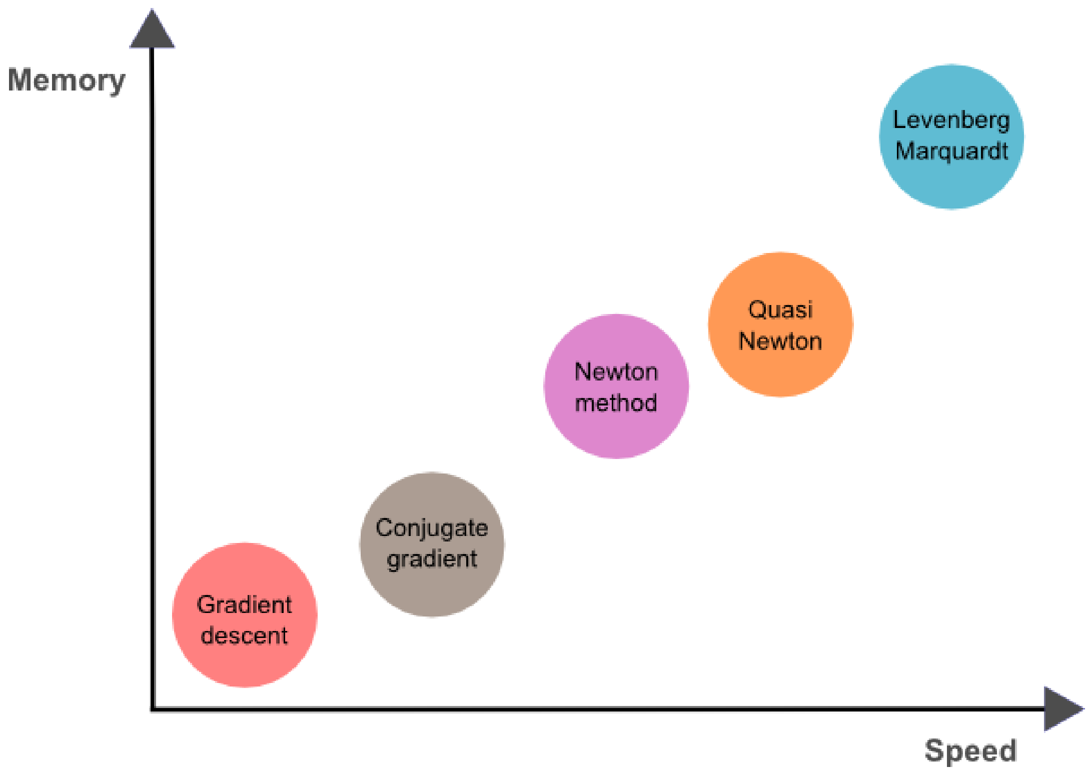
The picture below represents a state diagram for the training process of a neural network with the Levenberg-Marquardt algorithm. The first step is to calculate the loss, the gradient and the Hessian approximation. Then the damping parameter is adjusted so as to reduce the loss at each iteration.



As we have seen the Levenberg-Marquardt algorithm is a method tailored for functions of the type sum-of-squared-error. That makes it to be very fast when training neural networks measured on that kind of errors. However, this algorithm has some drawbacks. The first one is that it cannot be applied to functions such as the root mean squared error or the cross entropy error. Also, it is not compatible with regularization terms. Finally, for very big data sets and neural networks, the Jacobian matrix becomes huge, and therefore it requires a lot of memory. Therefore, the Levenberg-Marquardt algorithm is not recommended when we have big data sets and/or neural networks.

Memory and speed comparison

discussed in this post. As we can see, the slowest training algorithm is usually gradient descent, but it is the one requiring less memory. On the contrary, the fastest one might be the Levenberg-Marquardt algorithm, but usually requires a lot of memory. A good compromise might be the quasi-Newton method.



To conclude, if our neural networks has many thousands of parameters we can use gradient descent or conjugate gradient, in order to save memory. If we have many neural networks to train with just a few thousands of instances and a few hundreds of parameters, the best choice might be the Levenberg-Marquardt algorithm. In the rest of situations, the quasi-Newton method will work well.

Latest posts

- 6 testing methods for binary classification
- Perceptron: the main component of neural networks
- Improving risk assessment using Advanced Analytics

- 3 methods to deal with outliers
- What is advanced analytics?
- Genetic algorithms for feature selection in Data Analytics