Variations on Backpropagation

Backprop Variations

- Heuristic Modifications
  - Momentum
  - Variable Learning Rate
  - Quickprop

- Standard Numerical Optimization
  - Conjugate Gradient
  - Newton’s Method (Levenberg-Marquardt)

Error Surface Example

Network Architecture

Nominal Function

Parameter Values

Squared Error vs. $w^{1,1}$ and $w^{2,1}$

Squared Error vs. $w^{1,1}$ and $b^{1}$

Squared Error vs. $b^{1}$ and $b^{2}$
Convergence Example

Learning Rate Too Large

Momentum Backpropagation

Steepest Descent Backpropagation (SDBP)
\[
\|W^i\| = -\|\varepsilon\|a_{i-1}^{-1}f \\
\|b^i\| = -\|\varepsilon\|
\]

Momentum Backpropagation (MOBP)
\[
\|W^i\| = \|W^i\| - \|D_s^i\| + (1 - \|D_s^i\|)g_i \\
\|b^i\| = \|b^i\| - \|D_s^i\| + (1 - \|D_s^i\|)g_i
\]

\[g = 0.8\]

Variable Learning Rate (VLBP)

• If the squared error (over the entire training set) increases by more than some set percentage \(\varepsilon\) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor \(1 > \varepsilon > 0\), and the momentum coefficient \(g\) is set to zero.

• If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor \(\varepsilon > 1\). If \(g\) has been previously set to zero, it is reset to its original value.

• If the squared error increases by less than \(\varepsilon\), then the weight update is accepted, but the learning rate and the momentum coefficient are unchanged.

Example

The remaining slides describe two methods for accelerating backpropagation

• Conjugate Gradient method
• Levenberg-Marquardt method
Steepest Descent (Gradient Descent) Review

Basic Optimization Algorithm

\[ x_{k+1} = x_k + \alpha_k p_k \]

or

\[ \Delta x = (x_{k+1} - x_k) = \alpha_k p_k \]

- \( p_k \) - Search Direction
- \( \alpha_k \) - Learning Rate

Example for an Analytic Function

\[ F(x) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_3 \]

\[ x_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \]

\[ \alpha = 0.1 \]

\[ \nabla F(x) = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \]

\[ p_0 = \nabla F(x)|_{x=x_0} = \begin{bmatrix} -0.5 \\ -0.2 \end{bmatrix} \]

\[ x_1 = x_0 - \alpha p_0 = \begin{bmatrix} 0.5 \\ 0.1 \\ 0.2 \end{bmatrix} \]

\[ x_2 = x_1 - \alpha p_1 = \begin{bmatrix} 0.3 \\ 0.1 \\ 0.2 \end{bmatrix} \]

Plot

Note: Lots of small steps

Conjugate Gradient Method
Minimizing Function Along a Line

Compute learning rate $\eta$ to minimize $F(x_k + \eta D_k)$, $D_k$ is any chosen line direction, e.g. $-g_k$ (negative gradient).

By Taylor expansion:
$$
\frac{d}{d\eta} F(x_k + \eta D_k) = \left[ \frac{d}{d\eta} F(x_k) \right]_{\eta=0} D_k + \left[ \frac{d^2}{d\eta^2} F(x_k) \right]_{\eta=0} (D_k)^2
$$

Set derivative to 0 and solve for $\eta$:
$$
\eta = \frac{-\left[ \frac{d}{d\eta} F(x_k) \right]_{\eta=0}}{\left[ \frac{d^2}{d\eta^2} F(x_k) \right]_{\eta=0}} = \frac{g_k^T D_k}{g_k^T D_k} \text{ is where derivative is 0}
$$

The analytic Hessian $A_k = \left[ \frac{d^2}{d\eta^2} F(x_k) \right]_{\eta=0}$.

Successive Line Minimizations with different directions

For Quadratic Functions

$$
\frac{d}{d\eta} F(x_k + \eta D_k) = \frac{d}{d\eta} F(x_k + \eta D_k) = \left[ \frac{d}{d\eta} F(x_k) \right]_{\eta=0} D_k + \left[ \frac{d^2}{d\eta^2} F(x_k) \right]_{\eta=0} (D_k)^2
$$

The change in the gradient at iteration $k$ is
$$
\left[ \frac{d}{d\eta} F(x_k + \eta D_k) \right]_{\eta=0} = \left[ \frac{d}{d\eta} F(x_k + \eta D_k) \right]_{\eta=0} = -A_k x_k + d
$$

where
$$
A_k = \left[ \frac{d^2}{d\eta^2} F(x_k) \right]_{\eta=0}
$$

The conjugacy conditions can be rewritten
$$
\eta^T A_k p_j = 0 \quad \text{for } j \neq k
$$

This does not require knowledge of the Hessian matrix.

Example

$$
F(x) = \frac{1}{2} x^T A x + b^T x + \frac{1}{2}
$$

$$
\nabla F(x_k) = [\frac{\partial F(x_k)}{\partial x_1}, \frac{\partial F(x_k)}{\partial x_2}] = [2x_1 + 2b_1, 2x_2 + 2b_2]
$$

$$
p_k = -\nabla F(x_k) = \frac{\partial F(x_k)}{\partial x} = [0.5, 0.5, 0.2, 0.2]
$$

Conjugate Vectors

A set of vectors is mutually conjugate with respect to a positive definite Hessian matrix $A$ if
$$
\nabla F(x)^T A p_j = 0 \quad \text{for } j \neq k
$$

One set of conjugate vectors consists of the eigenvectors of $A$.

$$
\lambda \mathbf{A} \mathbf{x}_j = \frac{\partial F(x)}{\partial x} \mathbf{x}_j = 0 \quad \text{for } j \neq k
$$

(The eigenvectors of symmetric matrices are orthogonal.)

Forming Conjugate Directions

Choose the initial search direction as the negative of the gradient:
$$
p_0 = -g_0
$$

Choose subsequent search directions to be conjugate:
$$
p_k = -g_k + \eta_k p_{k-1}
$$

where $\eta_k$ is chosen by one of these formulas:
$$
\eta_k = \frac{\|g_k\|^2}{\nabla F(x_k)^T A \nabla F(x_k)} \text{ or } \eta_k = \frac{-\nabla F(x_k)^T A \nabla F(x_k)}{\nabla F(x_k)^T A \nabla F(x_k)} \text{ or } \eta_k = \frac{\|g_k\|^2}{\nabla F(x_k)^T A \nabla F(x_k)}
$$

Hestenes & Steifel
Fletcher–Reeves
Polak & Ribiere

9-1#
Conjugate Gradient algorithm

- The first search direction is the negative of the gradient.
- \( p_1 = -g_0 \)
- Select the learning rate to minimize along the line.
  \[ \eta_k = -\frac{\nabla^2 F(x_k)}{\nabla^2 F(x_k)} p_k \]
  (For quadratic functions only.)
- Select the next search direction using
  \[ p_k = \theta g_k + \eta_k p_{k-1} \]
- If the algorithm has not converged, return to second step.
- If the function were quadratic, it would be minimized in \( n \) steps.

Example

\[ F(x) = \frac{1}{2} x^T A x + b^T x \]
\[ x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]
\[ A = \begin{bmatrix} 2 & 2 \\ 2 & 0 \end{bmatrix} \]
\[ b = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \]
\[ x_1 = x_0 - \eta \]
\[ \eta = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.1 \end{bmatrix} \]

Plots

Conjugate Gradient

Steepest Descent

Numerical Line Minimization

- First locate an interval containing the minimum.
- Then reduce the interval’s width successively, until the interval is sufficiently small that we are close enough to the minimum.
Interval Location to Bracket Minimum

Evaluate \( F(x_k + n \delta_k) \) for a doubling ...

Interval Reduction

Dividing in two parts

Dividing in three parts

Golden Section (Fibonacci) Search

\[ t = 0.618 \]

Set \( c_1 = a_1 + (1 - t)(b_1 - a_1) \), \( F_a = F(c_1) \)

For \( k = 2, 3, \ldots \) repeat

If \( F_a < F_c \) then

Set \( a_{k+1} = a_k \); \( b_{k+1} = d_k \); \( d_{k+1} = c_k \)

\( F_a = F_c \); \( F_c = F(d_{k+1}) \)

else

Set \( a_{k+1} = c_k \); \( b_{k+1} = b_k \); \( c_{k+1} = d_k \)

\( d_{k+1} = b_{k+1} - (1 - t)(b_{k+1} - a_{k+1}) \)

\( F_a = F_d \); \( F_d = F(c_{k+1}) \)

end

end until \( b_{k+1} - a_{k+1} < \text{tol} \)

Conjugate Gradient BP (CGBP)

Intermediate Steps

Complete Trajectory

Levenberg-Marquardt Method

(Blends Newton’s method with steepest descent)

Probably the fastest known method for training (but storage intensive)

Levenberg-Marquardt Method

\[ x_{k+1} = x_k - A_k^{-1} \nabla \]

\[ A_k = \left( \frac{\partial F}{\partial x} \right)_k \]

\[ \nabla = \left( \frac{\partial F}{\partial x} \right)_k \]

If the performance index is a sum of squares function:

\[ F(x) = \sum_{i=1}^{N} \epsilon_i^2(x) = \sum_{i=1}^{N} \epsilon_i^2(x) \]

then the \( j \)th element of the gradient is

\[ \left( \frac{\partial F}{\partial x} \right)_j = \sum_{i=1}^{N} \frac{\partial \epsilon_i^2(x)}{\partial x_j} \]
Newton’s Method

\[ f(x_{k+1}) = f(x_k) + \nabla f(x_k) + \frac{1}{2} J^T(x_k) \Delta x_k \]

Take the gradient of this second-order approximation and set it equal to zero to find the stationary point:

\[ \Delta x_k = -J(x_k)^T \nabla^2 f(x_k) \]

\[ x_{k+1} = x_k - \Delta x_k \]

Example

\[ f(k) = x_k^2 + 2x_{k+1} + 2x_k^2 + x_k \]

Initial Point: \( x_0 = \frac{1}{2} \)

\[ \nabla f(x_k) = \begin{bmatrix} \frac{2x_k}{x_k} \ 2x_{k+1} + 1 \end{bmatrix} \]

\[ \Delta x_k = \left( \begin{bmatrix} \frac{2x_k}{x_k} \\ 2x_{k+1} + 1 \end{bmatrix} \right)^T \nabla^2 f(x_k) \]

\[ A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \]

\[ \Delta x_k = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \]

Stationary Point: \( x = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \)

Non-Quadratic Example

\[ f(k) = (x_2 - x_1)^3 + 8x_1x_2 - x_1 + x_2 \]

Stationary Points:

\[ x^1 = \begin{bmatrix} -0.42 \\ 0.42 \end{bmatrix} \]

\[ x^2 = \begin{bmatrix} 0.13 \\ 0.13 \end{bmatrix} \]

\[ x^3 = \begin{bmatrix} 0.55 \\ 0.55 \end{bmatrix} \]

Matrix Form

The gradient can be written in matrix form as a matrix-vector product:

\[ \nabla f(x) = J(x) \begin{bmatrix} \frac{\partial f_1}{\partial x} \\ \frac{\partial f_2}{\partial x} \end{bmatrix} \]

where \( J \) is the Jacobian matrix:

\[ J(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \]
Hessian represents Curvature
Express in terms of Jacobian:
\[
[D^2 F(x)]_{ij} = \sum_{v=1}^{N} \frac{\partial^2 F(x)}{\partial x_i \partial x_j} = \sum_{v=1}^{N} \left( \frac{\partial^2 F(x)}{\partial x_i \partial x_j} + \frac{\partial^2 F(x)}{\partial x_j \partial x_i} \right)
\]
\[
[D^2 F(x)] = 2J'F(x)J(x) + 2S(x)
\]

Levenberg-Marquardt
Gauss-Newton approximates the Hessian by:
\[
H = J \cdot J
\]
This matrix may be singular, but can be made invertible as follows:
\[
G = H + I
\]
If the eigenvalues and eigenvectors of \( H \) are:
\[
\lambda_i, \ldots, \lambda_n \quad \text{and} \quad e_i, \ldots, e_n
\]
then
\[
G e_i = (H + I) e_i = \text{Eigenvectors of } G
\]
\[
g_i = \text{Eigenvalues of } G
\]
\[
x_{k+1} = x_k - \left[ J'(x_k) J(x_k) \right] ^{-1} J'(x_k) w(x_k)
\]

Application to Multilayer Network
The performance index for the multilayer network is:
\[
F(x) = \sum_{i=1}^{Q} (a_i - y_i) (a_i - y_i) = \sum_{i=1}^{Q} e_i e_i = \sum_{i=1}^{Q} \left( \epsilon_i \right)^2
\]
The error vector is:
\[
\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \ldots \\ \epsilon_n \end{bmatrix}
\]
The parameter vector is:
\[
x = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_m \end{bmatrix}
\]
The dimensions of the two vectors are:
\[
N = Q \times m
\]
\[
n = \sum_{i=1}^{Q} (Q+1) + \sum_{i=1}^{Q} (Q+1) + \ldots + \sum_{i=1}^{Q} (Q+1)
\]

Gauss-Newton Method
Approximate the Hessian matrix as:
\[
[D^2 F(x)] = J'(x) J(x)
\]
Newton's method becomes:
\[
x_{k+1} = x_k - \left[ J'(x_k) J(x_k) \right] ^{-1} J'(x_k) w(x_k)
\]
\[
x_{k+1} = x_k - \left[ J'(x_k) J(x_k) \right] ^{-1} J'(x_k) w(x_k)
\]

Adjustment of \([k]\)
As \([k] = 0 \), LM becomes Gauss-Newton.
\[
x_{k+1} = x_k - \left[ J'(x_k) J(x_k) \right] ^{-1} J'(x_k) w(x_k)
\]
As \([k] > 0 \), LM becomes Steepest Descent with small learning rate.
\[
x_{k+1} = x_k - \frac{\partial^2 F(x)}{\partial x_i \partial x_j} = x_k - \frac{\partial^2 F(x)}{\partial x_i \partial x_j}
\]
Steepest Descent is a default position: Begin with a small \([k]\) to use Gauss-Newton. If a step does not yield a smaller \( F(x) \), then repeat the step with an increased \([k]\) until \( F(x) \) is decreased. \( F(x) \) must decrease eventually, since we will eventually be taking a very small step in the steepest descent direction.

Jacobian Matrix for Levenberg-Marquardt
\[
J(x) = \begin{bmatrix}
\frac{\partial F(x)}{\partial x_1} & \frac{\partial F(x)}{\partial x_2} & \ldots & \frac{\partial F(x)}{\partial x_m}
\end{bmatrix}
\]
\[
M \text{ rows for every input sample}
\]
Computing the Jacobian

Steepest descent computes terms of the form:
\[
\frac{\partial f(x)}{\partial x_i} \cdot \delta x_i
\]
using the chain rule:
\[
\frac{\partial f}{\partial x_j} = \frac{\partial f}{\partial \delta x_i} \cdot \frac{\partial \delta x_i}{\partial x_j}
\]
where the sensitivity
\[
\delta x_i = x_i - \tilde{x}_i
\]
is computed using backpropagation.

For the Jacobian we need to compute terms of the form:
\[
[J]_{j,i} = \frac{\partial s_j}{\partial x_i}
\]

Marquardt Sensitivity

If we define a Marquardt sensitivity \( q \) is the sample’s index:
\[
\frac{\partial f^q}{\partial x_i} = \frac{\partial e_q}{\partial x_i}
\]
we can compute the Jacobian as follows:

\[
\begin{align*}
[J]_{j,i} &= \frac{\partial f^q}{\partial x_i} \\
&= \frac{\partial e_q}{\partial x_i} \\
&= \frac{\partial e_q}{\partial w} \cdot \frac{\partial w}{\partial x_i} + \frac{\partial e_q}{\partial b} \cdot \frac{\partial b}{\partial x_i}
\end{align*}
\]

Levenberg–Marquardt Backpropagation

- Present all inputs to the network and compute the corresponding network outputs and the errors. Compute the sum of squared errors over all inputs.
- Compute the Jacobian matrix. Calculate the sensitivities with the backpropagation algorithm, after initializing. Augment the individual matrices into the Marquardt sensitivities. Compute the elements of the Jacobian matrix.
- Solve to obtain the change in the weights.
- Recompute the sum of squared errors with the new weights. If this new sum of squares is smaller than that computed in step 1, then divide \[b\] by \[a\] and go back to step 1. If the sum of squares is not reduced, then multiply \[b\] by \[a\] and go back to step 3.

Example LMBP Step

LMBP Trajectory
Quickprop
Scott Fahlman, CMU

- This is an optimization of backpropagation based on Newton’s method.
- It is applicable when, between two steps, the gradient has decreased in magnitude and has changed sign.
- Then a parabolic estimate of the MSE is used to determine the weights for the next step.

Quickprop, step k, in 1 dimension

Quickprop

- Assume a parabola:
  \[ J(w) = aw^2 + bw + c \]
- First derivative is a line:
  \[ \frac{dJ}{dw} = 2aw + b \]
abbreviate \( \frac{dJ}{dw} \) as \( J'(w) \).
- To find: value of \( w(k+1) \) such that \( J'(w(k+1)) = 0 \).
- We have:
  \[ J'(w(k)) = 2aw(k) + b \]
  \[ J'(w(k-1)) = 2aw(k-1) + b \]
- Solving for \( a \) and \( b \) in terms of the other quantities:
  \[ 2a = \frac{J'(w(k))-J'(w(k-1))}{w(k)-w(k-1)} \]
  \[ b = J'(w(k)) - \frac{(J'(w(k))-J'(w(k-1)))(w(k)-w(k-1))}{w(k)-w(k-1)} \]
where \( J'(w(k)) = w(k) \cdot J'(w(k)) \)

Quickprop

- Set \( J'(w(k+1)) = 0 \), since we are looking for the parabolic minimum at the next step.
- Then \( 2aw(k+1) + b = 0 \), i.e. \( w(k+1) = -b/(2a) \).
- Substituting in previous equations, we get
  \[ w(k+1) = w(k) + \frac{J'(w(k)) [w(k-1)]}{J'(w(k))-J'(w(k-1))] \]
as the choice for \( w(k+1) \).

9-#