Neural Networks

Assignment 5

Ravikiran Bhat
Rubanraj Ravichandran
Ramesh Kumar

November 11, 2017

1 Summary

1.1 Single-Layer Perceptrons

- Poineering contributions in neural network
  - McCulloch and Pitts introduced idea of neural networks as computing machines
  - Hebb introduced first rule of self-organized learning
  - Rosenblatt proposed model of perceptron for supervised learning
- **Rosenblatt's perceptron**
  - Perceptron is simplest form of neural network used for classification of linearly separable patterns
  - Consists of single neuron with adjustable weights and bias
  - Rosenblatt introduced perceptron convergence theorem that proved, if patterns/vectors used for training the network are linearly separable, then perceptron always converges and hyperplane can easily separate these patterns
  - Single neuron corresponds to the adaptive filter such as ADALINE(adaptive linear neuron)

1.2 Adaptive Filtering Problem

- Adaptive filtering operation consists of two continuous processes
  - Filtering process : Involves the computation of two signals
    - Output is produced based on input vectors
    - Error signal obtained by comparing current output to the desired output
  - Adaptive process : Involves weights adjustment based on error signal
- Combination of these two processes produce feedback loop acting around the neuron

1.3 Unconstrained Optimization Techniques

- It is a way of adjusting weight vector w of an adaptive filter algorithm to minimize the cost function with respect of weight vector
- Class of unconstrained optimization algorithms is based on idea of local iterative descent
  - Start with a initial guess denoted by w(0)
  - Generate a sequence of weight vectors w(1), w(2), ..., such that cost function is reduced at each iteration of the algorithm

Three unconstrained optimization methods that are based on idea of local iterative descent are:

1.3.1 Steepest Descent

- Successive adjustments of the weight vector w are in the direction of steepest descent; in a direction opposite to the gradient vector $\triangle\epsilon(w)$
- Steepest descent algorithm is formally described by:

  $$w(n+1) = w(n) - \eta g(n)$$

  where η is positive constant called step-size or learning parameter and g(n) is the gradient vector evaluated at the point w(n)
- Weight is adjusted as :

  $$\triangle w(n) = w(n+1) - w(n) = -\eta g(n)$$

- Learning-rate parameter has good influence on convergence behavior
  - When η is small, transient response is over-damped, and trajectory traced by w(n) follows a smooth path in the W-plane
  - When η is large, transient response is over-damped, and trajectory traced by w(n) follows a zigzag(oscillatory) path in the W-plane
  - When it exceeds than certain threshold, algorithm becomes unstable(and it diverges)

1.3.2 Newton's Method

- Basic idea of this method is to minimize the quadratic approximation of the cost function $\epsilon(w)$ around the current point w(n)
- second-order Taylor series expansion of the cost function is used around the point w(n)
- Generally, this method converges quickly asymptotically and does not exhibit the zigzagging behavior, unlike steepest descent method
- However, for newton's method to work
  - Hessian H(n) should be positive definite matrix for all iteration n
  - Unfortunately, in general there is no guarantee that H(n) is positive definite
  - If the Hessian H(n) is not positive definite, modifications of Newton's method is necessary

1.3.3 Gauss-Newton Method

- This method is applicable to a cost function that is expressed as sum of error squares(i.e. Special case of Newton) :

$$\epsilon(w) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

- Weight update is given as:

$$w(n+1) = w(n) - (J^T(n)J(n))^{-1}J^T(n)e(n)$$

- Gauss-Newton method only requires the Jacobian matrix of the error vector e(n), rather than knowledge of Hessian matrix of the cost function as in Gauss method
- However, for this method, iteration should be computable and the matrix product $J^T(n)J(n)$ must be nonsingular. In other words, n rows of J(n) must be linearly independent
- Unfortunately, rows of J(n) are always linearly independent, therefore, to get rid of this deficiency we add diagonal matrix δI to the matrix $J^T(n)J(n)$

- The parameter ρ is a small positive constant chosen to ensure that

$$J^T(n)J(n) + \rho I$$

is positive definite for all n

- On this basis, Gauss-Newton method is slightly modified as :

$$w(n+1) = w(n) - (J^T(n)J(n) + \delta I)^{-1} J^T(n)e(n)$$

1.4 Linear Least-Squares Filter

- Linear least-squares filter has two distinctive characteristics
  - This is one of application of Gauss-Newton method
  - Single neuron is linear
  - Cost function $\epsilon(w)$ used to design the filter consists of the sum of error squares
  - Error vector e(n) is given as:

    $$e(n) = d(n) - X(n)w(n)$$

    where d(n) is desired vector of n-by-1 and X(n) is input matrix of n-by-m
  - Differentiating above equation, we get:

    $$\triangle e(n) = -X^T(n)$$

  - Jacobian of e(n) is:

    $$J(n) = -X(n)$$

  - Apply Gauss Newton, we get:

    $$w(n+1) = X^+(n)d(n)$$

    where X^+ is pseudo-inverse

1.5 Least-Mean-Square Algorithm

- Least-mean-square(LMS) algorithm is based on use of instantaneous values for cost function:

$$\epsilon(w) = \frac{1}{2}e^2(n)$$

- LMS algorithm operates with a linear neuron, error signal is expressed as :

$$e(n) = d(n) - x^T(n)w(n)$$

- Weight update is given as :

$$\hat{w}(n+1) = \hat{w}(n) + \eta x(n)e(n)$$

  where η is learning-rate parameter and $\hat{w}$(n) is estimate of the weight vector

- In steepest descent algorithm weight vector w(n) follows a well-defined trajectory in weight space for prescribed n. While, in LMS algorithm, weight vector $\hat{w}(n)$ traces a random trajectory. Due to this reason, LMS algorithm is also called as Stochastic gradient algorithm

1.5.1 Convergence Considerations of the LMS Algorithm

- From practical point of view, first criterion for convergence of the LMS algorithm is convergence in the mean square :

$$E[e^2(n)] \longrightarrow constant As n \longrightarrow \inf$$

- LMS is convergent in the mean square provided that η satisfies the condition

$$0 < \eta < \frac{2}{\lambda_{max}}$$

  where λ_{max} is the largest eigenvalue of correlation matrix R

- In typical applications, knowledge of λ_{max} is not available. Therefore, we use trace of R_x as a conservative estimate for λ_{max}. Equation can be re-written as:

$$0 < \eta < \frac{2}{tr[R_x]}$$

1.5.2 Virtues and Limitations of the LMS Algorithm

- One of important virtue of the LMS algorithm is its simplicity, and since it is model independent and therefore robust; that means small model uncertainty and small disturbances can only result in small estimation errors
- Primary limitations of LMS algorithm are its slow rate of convergence and sensitivity to variations in the eigen structure of the input
- Mostly these algorithms require number of iterations equal to about 10 times the dimensionality of the input space for it to reach a steady-state condition
- Hessian matrix H is defined as the second derivative of the cost function $\epsilon(w)$ with respect to w, which is equal to R_x

2 Exercise 3.1

Explore the method of steepest decent involving a single weight w by considering the following cost function.
$\xi(\omega) = \frac{1}{2}\sigma^2 - r_{xd}\omega + \frac{1}{2}r_x\omega^2$

The gradient vector $g(n) = -r_{xd} + r_x\omega$

The cost function $\xi(\omega)(n+1)$ by the steepest descent algorithm is given by:
$\xi(\omega(n+1)) = \xi(\omega(n)) - \eta g(n)^2$.

After substituting the value of $g(n)$ in the above cost function equation, we get
$\xi(\omega(n+1)) = \xi(\omega(n)) - \eta(-r_{xd} + r_x\omega(n))^2$
$= (\frac{1}{2}\sigma^2 - r_{xd}\omega(n) + \frac{1}{2}r_x\omega(n)^2) - \eta(-r_{xd} + r_x\omega(n))^2$
$=(\eta r_x^2 + \frac{1}{2}r_x)\omega(n)^2 - (2\eta r_{xd}r_x + r_{xd})\omega(n) + (\eta r_{xd}^2 + \frac{1}{2}\sigma^2)$

3 Exercise 3.2

This exercise is solved using ipython and ipython file has been attached along with the submission files. When we downloaded pdf copy from jupyter notebook, the matrix brackets were not showing in the pdf file. So, please check our ipython file for proper formatted result.

4 Exercise 3.4

This exercise is solved using ipython and ipython file has been attached along with the submission files.

5 Exercise 3.8

5.1 (3.8 a)

Cost function is given as :

$$J(w) = \frac{1}{2}E[(d(n) - x^T(n)w)^2]$$

Expanding square formula, we have:

$$J(w) = \frac{1}{2}E(d^2(n) - 2d(n)x^T(n)w + w^Tx^T(n)x(n)w)$$

Above equation can be simplified as :

$$J(w) = \frac{1}{2}E[d^2(n)] - E[d(n)x^T(n)w] + \frac{1}{2}E[w^Tx^T(n)x(n)w]$$

Since $R_x = E[x(n)x^T(n)]$, $r_xd^T = E[x(n)^Td(n)]$ and $\sigma^2 = E[d^2(n)]$
Therefore, above equation can be written as:

$$J(w) = \frac{1}{2}\sigma^2 - r_{xd}^Tw + \frac{1}{2}w^TR_xw$$

5.2 (3.8 b)

Since cost function is given as:

$$J(w) = \frac{1}{2}\sigma^2 - r_{xd}^Tw + \frac{1}{2}w^TR_xw$$

Gradient vector is first partial derivative of cost function with respect to w, therefore, it can be written as:

$$\frac{\partial J(w)}{\partial w} = 0 - r_xd + \frac{1}{2}2R_xw$$

Simplify it, we have:

$$g = -r_x d + R_x w$$

Hessian matrix is defined as second partial derivative of cost function with respect to w, therefore, it can be computed as :

$$\frac{\partial J(w)}{\partial w} = -r_x d + R_x w$$

$$\frac{\partial J^2(w)}{\partial w^2} = 0 + R_x$$

Simplify it, we have

$$H = -R_x$$

5.3 (3.8 c)

In the newton method, update weight is given as :

$$w(n+1) = w(n) - H^{-1}(n)g(n) \tag{1}$$

Since Hessian matrix is equal to correlation matrix R_x. Therefore, above equation can be re-written as:

$$w(n+1) = w(n) - R^{-1}g(n) \tag{2}$$

When we replace g with instantaneous values:

$$g = -r_{xd} + R_x w$$

Substituting values of r_{xd} and R_x , we have

$$g = -x(n)d(n) + x(n)x^T(n)w(n)$$

Substitute g in equation(2) or LMS algorithm , we get :

$$w(n+1) = \hat{w}(n)\eta R^{-1}(-x(n)d(n) + x(n)x^T(n)w(n)) \tag{3}$$

Simplifying it, we get:

$$w(n+1) = \hat{w}(n) + \eta R^{-1}(x(n)d(n) - x(n)x^T(n)w(n)) \tag{4}$$