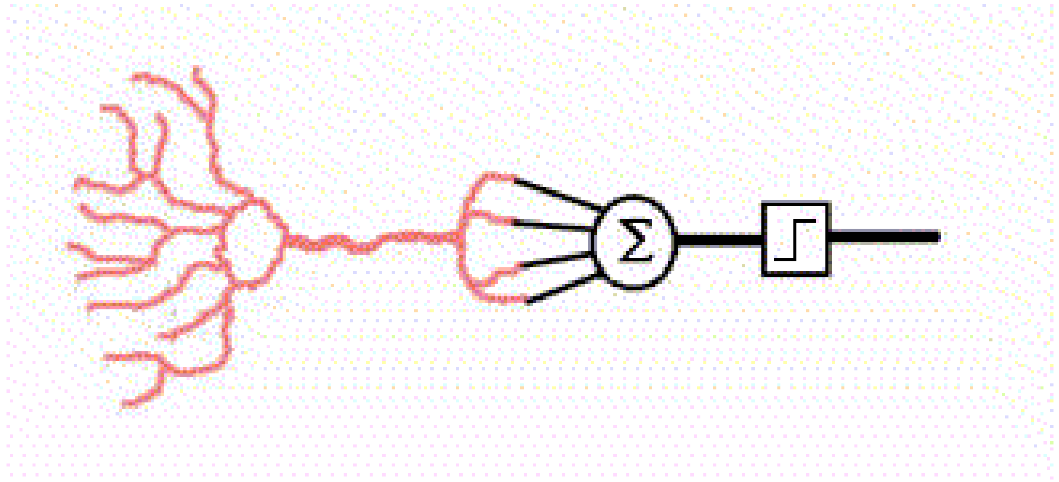


Neural Networks (CS010 805G02)

Mod 3 - Momentum,
Conjugate Gradient Learning,
Bias vs Variance

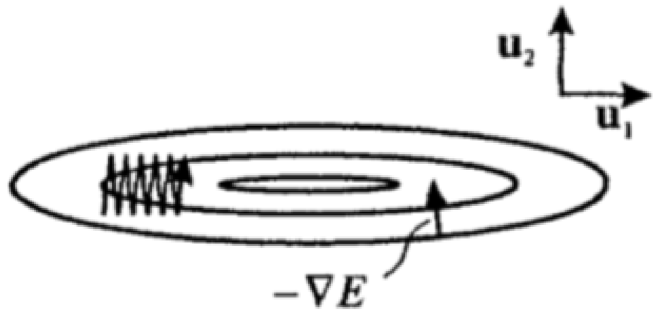
Shiney Thomas
AP,CSE,AJCE

Learning with Momentum

- One of the simplest network training algorithms, is gradient descent, sometimes also known as steepest descent.
- We start with some initial guess for the weight vector (which is often chosen at random) denoted by w^0.
- Then iteratively update the weight vector such that, at step **t**, we move a short distance in the direction of the greatest rate of decrease of the error, i.e. in the direction of the negative gradient, evaluated at w^t:

$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E|_{\mathbf{w}^{(\tau)}}$$

LEARNING WITH MOMENTUM



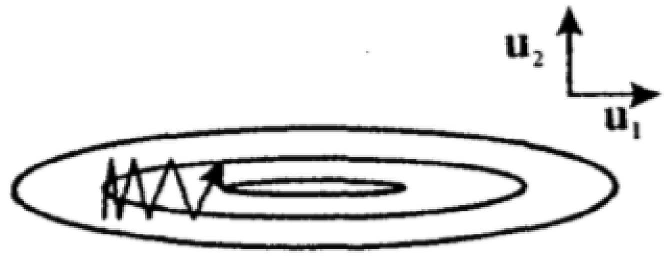
- One of the limitations of the gradient descent technique is the need to choose a suitable value for the learning rate parameter η.
- The problems with gradient descent do not stop there, however. Figure depicts the contours of E, for a hypothetical two-dimensional weight space, in which the curvature of E varies significantly with direction.
- At most points on the error surface, the local gradient does not point directly towards the minimum.
- Gradient descent then takes many small steps to reach the minimum, and is clearly a very inefficient procedure.

Learning with Momentum

- One very simple technique for dealing with the problem of widely differing eigenvalues is to add a momentum term to the gradient descent formula.
- This effectively adds inertia to the motion through weight space and smoothes out the oscillations depicted in previous Figure.
- The modified gradient descent formula is given by

$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E|_{\mathbf{w}^{(\tau)}} + \mu \Delta \mathbf{w}^{(\tau-1)}$$

  where μ (or α) is the momentum parameter
- Effect of adding momentum shows a rapid progress along the valley of the error function



Line Search

- The algorithms described involve taking a sequence of steps through weight space.
- Consider each of these steps in two parts.
  - First we must decide the direction in which to move, and
  - second, we must decide how far to move in that direction.
- With simple gradient descent, the direction of each step is given by the local negative gradient of the error function, and the step size is determined by an arbitrary learning rate parameter.
- We might expect that a better procedure would be to move along the direction of the negative gradient to find the point at which the error is minimized. More generally we can consider some search direction in weight space, and then find the minimum of the error function along that direction.
- This procedure is referred to as a line search, and it forms the basis for several algorithms which are considerably more powerful than gradient descent

Line Search

- Suppose that at step **t** in some algorithm the current weight vector is w^t, and we wish to consider a particular search direction d^t through weight space.
- The minimum along the search direction then gives the next value for the weight vector:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \lambda^{(\tau)}\mathbf{d}^{(\tau)}$$

where the parameter $\lambda^{(t)}$ is chosen to minimize

$$E(\lambda) = E(\mathbf{w}^{(\tau)} + \lambda\mathbf{d}^{(\tau)}).$$

Conjugate Gradients (from Wiki)

- In mathematics, the **conjugate gradient method** is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite.
- The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition. Large sparse systems often arise when numerically solving partial differential equations or optimization problems.
- The conjugate gradient method can also be used to solve unconstrained optimization problems such as energy minimization.

Conjugate Gradients

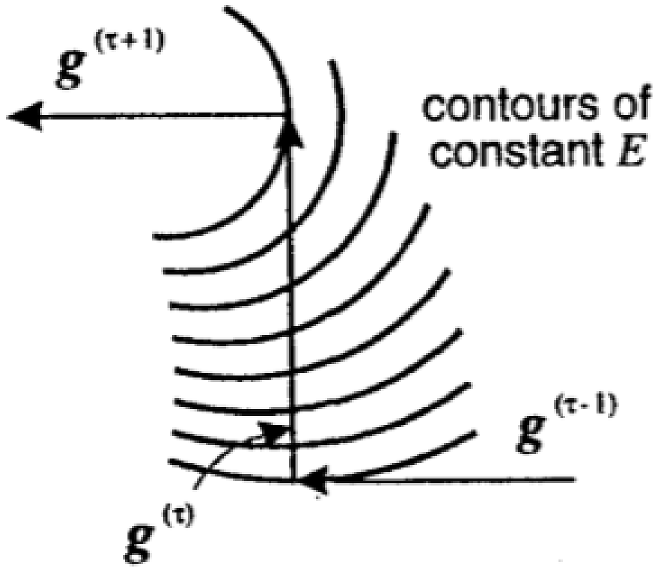
- To apply line search to the problem of error function minimization we need to choose a suitable search direction at each stage of the algorithm.
- However, the use of successive gradient vectors turns out in general not to represent the best choice of search direction.
- We note that at the minimum of the line search we have, (first derivative wrt step size)

$$\frac{\partial}{\partial \lambda} E(\mathbf{w}^{(\tau)} + \lambda \mathbf{d}^{(\tau)}) = 0$$

- which gives

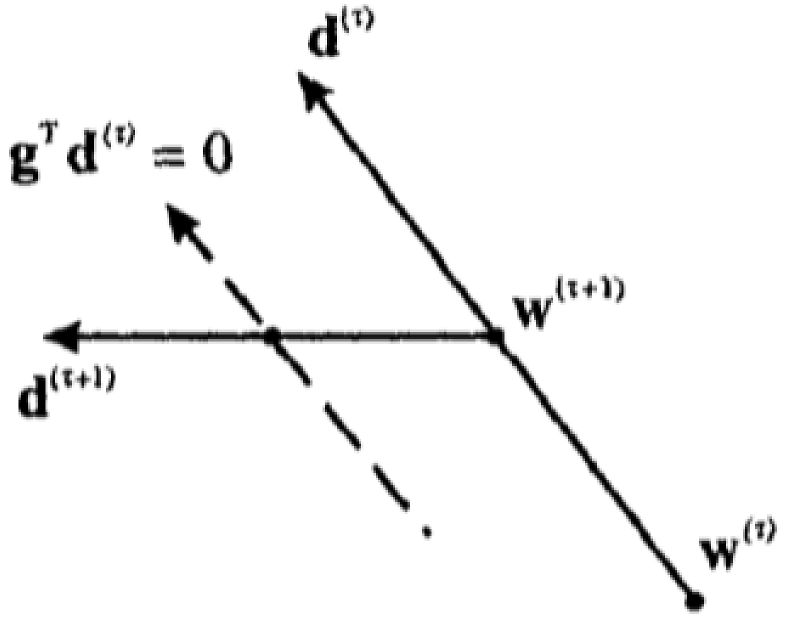
$$\mathbf{g}^{(\tau+1)\mathrm{T}} \mathbf{d}^{(\tau)} = 0 \qquad \text{where } \mathbf{g} \equiv \nabla E.$$

- After a line minimization, the new gradient is orthogonal to the line-search direction. Thus, if the search directions are always chosen to coincide with the negative gradients of the error function, as indicated here, then successive search directions will be orthogonal, and the error function minimization will typically proceed very slowly.



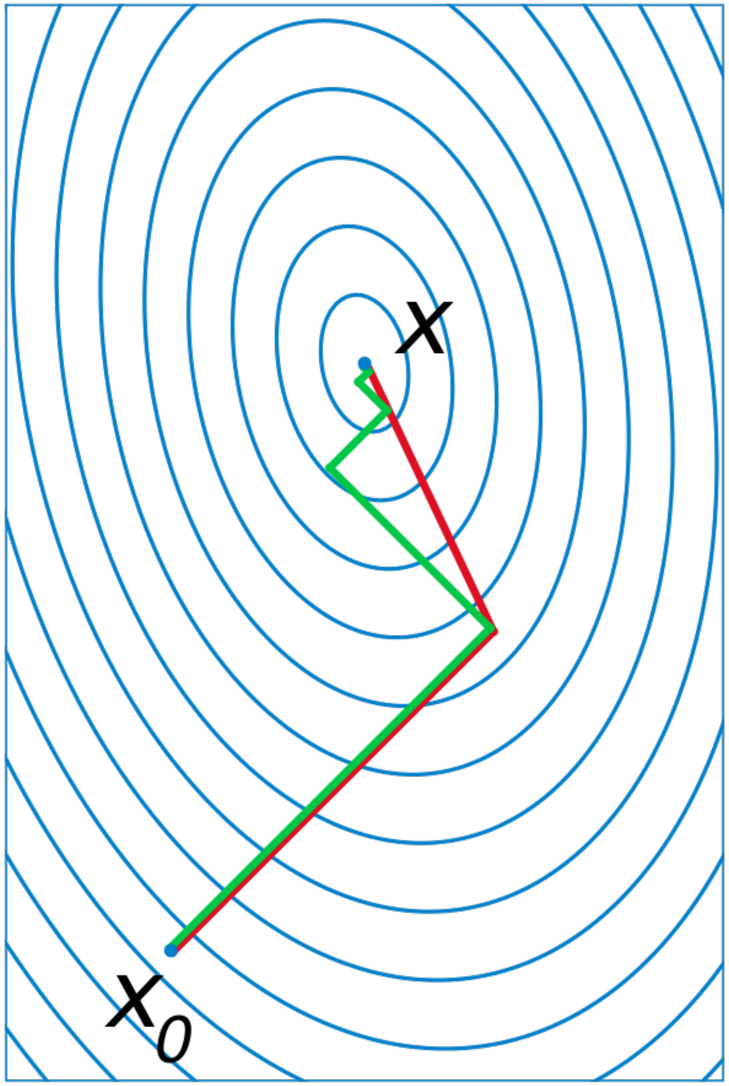
Conjugate Gradients

- The solution to this problem lies in choosing the successive search directions d^t such that, at each step of the algorithm, the component of the gradient parallel to the previous search direction, which has just been made zero, is unaltered (to lowest order).



Conjugate Gradients

- A comparison of the convergence of gradient descent with optimal step size (**in green**) and conjugate vector (**in red**) for minimizing a quadratic function associated with a given linear system. Conjugate gradient, assuming exact arithmetic, converges in at most *n* steps where *n* is the size of the matrix of the system (here *n*=2).



Conjugate Gradient Learning

- All of the conjugate gradient algorithms start out by searching in the steepest descent direction (negative of the gradient) on the first iteration. $\mathbf{d}_1 = -\mathbf{g}_1,$
- A line search is then performed to determine the optimal distance to move along the current search direction:

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j \mathbf{d}_j .$$

- Then the next search direction is determined so that it is conjugate to previous search directions.
- Then choosing each successive direction to be a linear combination of the current gradient and the previous search direction

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j .$$

Conjugate Gradient Learning

- The coefficients β_j can be found by imposing the conjugacy condition which gives

$$\beta_j = \frac{\mathbf{g}_{j+1}^T \mathbf{H} \mathbf{d}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j}.$$

H=Hessian matrix

- Three forms of representation:

1) Hestenes-Stiefel expression

$$\beta_j = \frac{\mathbf{g}_{j+1}^T (\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{d}_j^T (\mathbf{g}_{j+1} - \mathbf{g}_j)}$$

Conjugate Gradient Learning

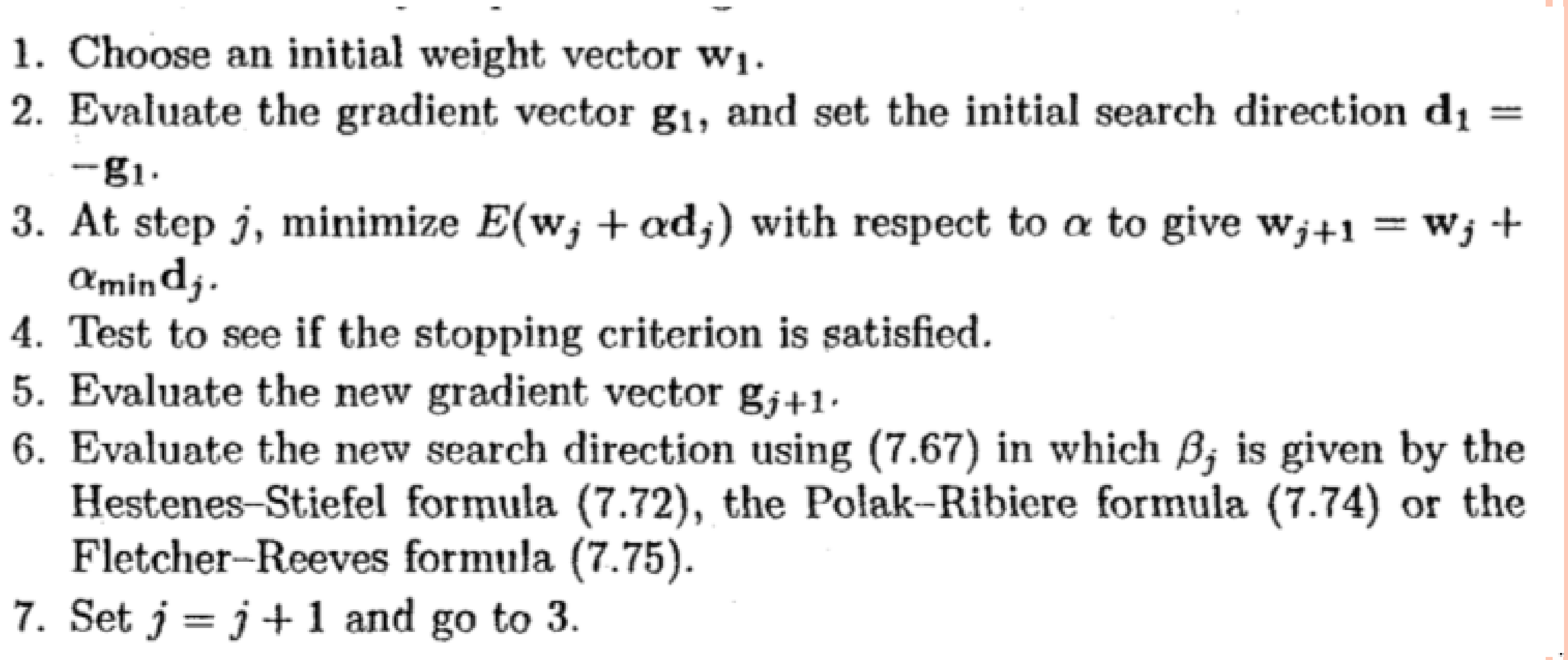
2) Polak-Ribiere form

$$\beta_j = \frac{\mathbf{g}_{j+1}^{\mathrm{T}}(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j^{\mathrm{T}}\mathbf{g}_j}.$$

3) Fletcher-Reeves form

$$\beta_j = \frac{\mathbf{g}_{j+1}^{\mathrm{T}}\mathbf{g}_{j+1}}{\mathbf{g}_j^{\mathrm{T}}\mathbf{g}_j}.$$

CG Algorithm



1. Choose an initial weight vector $\mathbf{w}_1$.
2. Evaluate the gradient vector $\mathbf{g}_1$, and set the initial search direction $\mathbf{d}_1 = -\mathbf{g}_1$.
3. At step j, minimize $E(\mathbf{w}_j + \alpha\mathbf{d}_j)$ with respect to α to give $\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_{\text{min}}\mathbf{d}_j$.
4. Test to see if the stopping criterion is satisfied.
5. Evaluate the new gradient vector $\mathbf{g}_{j+1}$.
6. Evaluate the new search direction using (7.67) in which β_j is given by the Hestenes–Stiefel formula (7.72), the Polak–Ribiere formula (7.74) or the Fletcher–Reeves formula (7.75).
7. Set $j = j + 1$ and go to 3.

Bias, Variance

- In probability theory and statistics, **variance** is the expectation of the squared deviation of a random variablefrom its mean. Informally, it measures how far a set of (random) numbers are spread out from their average value.

Bias, Variance

- Two common errors that we are going to look at are that of bias and Variance, and how a trade-off can be achieved between the two in order to generate a successful network model.
- How is this training data prepared? This is done by using a dataset for which the output of the algorithm is known.
- During the training stage, the algorithm analyzes the training data that is fed and produces patterns which are captured within an inferred function. This inferred function, which is derived after analysis of the training dataset, is the model that would be further used to map new examples.
- An ideal model generated from this training data should be able to **generalize** well. This means, it should learn from the training data and should correctly predict or classify data within any new problem instance.

BIAS,VARIANCE

- In general, the more complex the model is, the better it classifies the training data. However, if the model is too complex i.e it will pick up random features i.e. noise in the training data, this is the case of **overfitting** i.e. the model is said to overfit .
- On the other hand, if the model is not so complex, or missing out on important dynamics present within the data, then it is a case of **underfitting**.
- Both overfitting and underfitting are basically errors in the ML models or algorithms. Also, it is generally impossible to minimize both these errors at the same time and this leads to a condition called as the **Bias-Variance Tradeoff**.

Bias,Variance

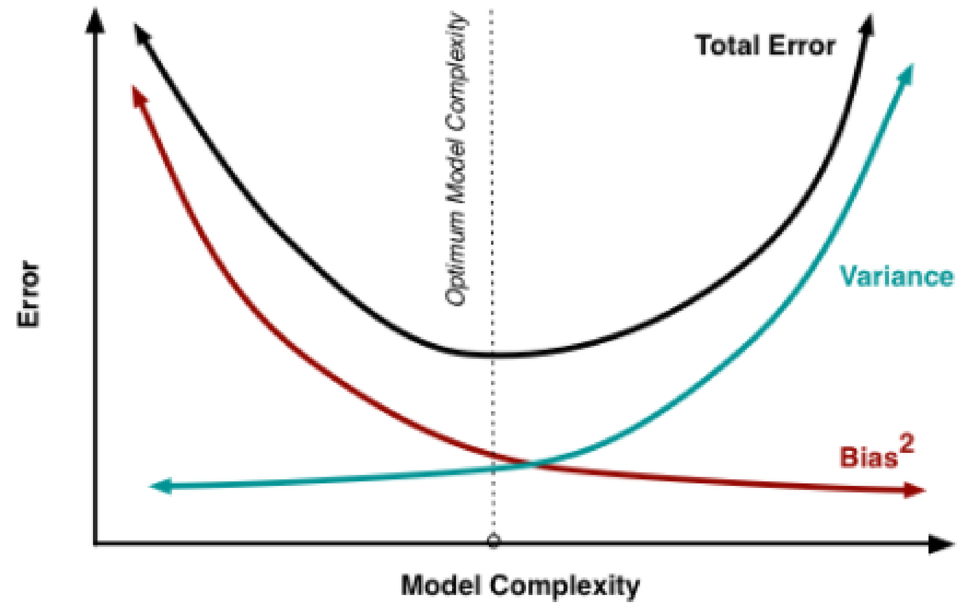


Fig. 6 Bias and variance contributing to total error.

More on Bias ,Variance:
Neural_Networks_for_Pattern_Recognition_Christopher_Bishop.pdf
https://www.youtube.com/watch?v=sgN6Flwejus