Optimization Methods	Date: *20/03/2018*
Instructor: *Sujit Prakash Gujar*	Scribes: Nitin Nilesh Shanmukh Alle Rama Dasu Mallelli

Lecture 18: Application of Gradient Descent: Neural Networks

1 Recap

1.1 Newton-Raphson Method

Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous, differentiable function. Our aim is to minimize $f(x)$ which can be written as:

$$\min_{x \in \mathbb{R}^n} f(x)$$

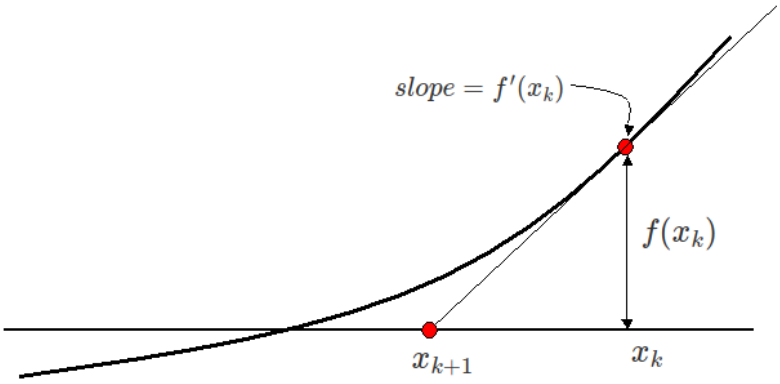

Figure 1: Graphical representation of Newton-Raphson Method

The equation of tangent at x_k is given by:

$$f(x) = f(x_k) + (x - x_k)f'(x_k)$$

We can find the root of this equation by setting $f(x) = 0$ and $x = x_{k+1}$ for our new approximation. By solving it, we get:

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$

1.1.1 Newton-Raphson method for $f : \mathbb{R}^n \to \mathbb{R}$

$$\boxed{x_{n+1} = x_n - \nabla^2 f(x_n)^{-1} \nabla f(x_n)}$$

where $\nabla^2 f(x_n)$ is the Hessian matrix and $\nabla f(x_n)$ is the gradient matrix

1.1.2 An Example:

Given $f(x,y) = x^2 + y^2$, find roots of $f(x) = 0$ with initial point $x_0 = \begin{bmatrix}1\\1\end{bmatrix}$

$$\nabla f(x,y) = \begin{bmatrix}2x\\2y\end{bmatrix}, \qquad \nabla f(x_0) = \begin{bmatrix}2\\2\end{bmatrix}$$

$$\nabla^2 f(x,y) = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}, \qquad \nabla^2 f(x,y)^{-1} = \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}$$

So according to the Newton-Raphson method, our update x_1 is:

$$x_1 = \begin{bmatrix}1\\1\end{bmatrix} - \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}$$

$$x_1 = \begin{bmatrix}1\\1\end{bmatrix} - \begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

If equation given is quadratic then Newton-Raphson will give the solution in one step.

1.1.3 Convergence of Newton-Raphson method

Suppose x_r is a root of $f(x) = 0$ and x_n is an estimate of x_r such that $|x_r - x_n| = \delta << 1$. Then by Taylor series expansion we have,

$$0 = f(x_r) = f(x_n + \delta) = f(x_n) + f'(x_n)(x_r - x_n) + \frac{f''(\zeta)}{2}(x_r - x_n)^2 \tag{1}$$

for some ζ between x_n and x_r.
By Newton-Raphson method, we know that,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

i.e.

$$f(x_n) = f'(x_n)(x_n - x_{n+1}) \tag{2}$$

using 2 in 1 we get,

$$0 = f'(x_n)(x_r - x_{n+1}) + \frac{f''(\zeta)}{2}(x_r - x_n)^2$$

$$say\, \varepsilon_k = (x_r - x_n),\ \varepsilon_{k+1} = x_r - x_{n+1}$$

where ε_k and ε_{k+1} denote the error in the solution at n^{th} and $(n+1)^{th}$ iterations.

$$\therefore \varepsilon_{k+1} = -\frac{f''(\zeta)}{2f'(x_n)} \approx \varepsilon_k^2$$

$$\Rightarrow \varepsilon_{k+1} \propto \varepsilon_k^2$$

$\therefore$ Newton-Raphson method is said to have quadratic convergence.

1.1.4 General form of descent methods

If x_k is our current point, d_k is the direction to move along and α_k is step size, using gradient descent, we move to our next point as such,

$$x_{k+1} = x_k + \alpha_k d_k$$

where $\alpha > 0$ and d_k satisfies $\nabla f(x_k)^T d_k < 0$
In general: $d_k = -D_k \nabla f(x_k)$, where D_k is Positive Definite.
So, we hope to get a reduction in $f(x)$ with the following approximation:

$$\boxed{f(x) = f(x_k) - \alpha \nabla f(x_k)^T D_k \nabla f(x_k)}$$

1.1.5 Variations of descent methods

In general, we can consider $d_k = -D_k \nabla f(x_k)$

- **Steepest Descent Method**

$$d_k = -\nabla f(x_k)$$

- **Newton Method**

$$d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

- **Diagonally Scaled Steepest Descent Method**

$$D_k = \begin{bmatrix} d_{k_1} & 0 & \dots & 0 & 0 \\ 0 & d_{k_2} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & d_{k_{n-1}} & 0 \\ 0 & 0 & \dots & 0 & d_{k_n} \end{bmatrix}$$

  where,

$$d_{k_i} = \left(\frac{\partial^2 f(x_k)}{(\partial x_i)^2} \right)^{-1}$$

- **Discretized Newton's Method**

$$D_k = (H(x_k))^{-1}$$

  Here, $H(x_k)$ is an approximation of $\nabla^2 f(x_k)$ using finite difference method and H is Positive Definite.

- **Modified Newton's Method**
  Newton-Raphson Method depend upon the computation of inverse of the Hessian matrix in every step and computation of inverse of matrix is computationally expensive. So to reduce computational complexity use same D_k for p steps where $p > 1$.

2 Application of Gradient Descent: Neural Networks

2.1 Introduction

Gradient descent methods are heavily used in machine learning/deep learning for solving problems related to regression, vision, natural language processing etc. One specific type of machine learning algorithm called Neural Networks uses gradient descent methods to solve different type of problems in Machine Learning. In this section we'll see how this works.

2.2 Neural Network

Consider a Neural Network having 3 layers: 1 input layer, 1 hidden layer, 1 output layer. Although a neural network can have l number of layers but for the sake of understanding we are using 3 layer neural network.
Notations are as follows(Please refer to the figure 2):

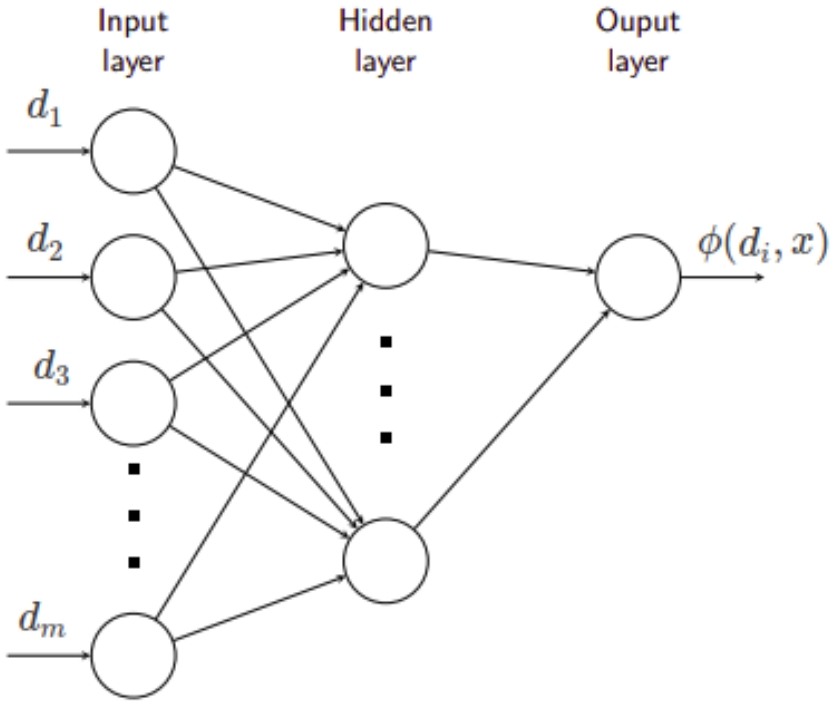

Figure 2: A three layer Neural Network

- n: Number of learnable parameters which represents weights of layers.
- x: Weights of different layers stored in an array $x = [x_1, x_2, ..., x_n]^T$
- d: Input data point of n dimensions. $d = [d_1, d_2, ..., d_n]^T$
- m: Number of input data points each having n dimensions.
- y_i: Class label for data point d_i. For example, class label for two class classification problem would be $y_i = +1$ or $y_i = -1$.
- $\phi(d_i, x)$: Classification result of our neural network for data point d_i on weights x.
  $\phi(d_i, x) \in \{+1, -1\}$
- $g_i(x)$: Error of data point d_i on weight x.
  $g_i(x) = \phi(d_i, x) - y_i$
  $g_i(x) : \mathbb{R}^n \to \mathbb{R}$
- $g(x)$: Error for all data points. $g(x) = [g_1(x), g_2(x), ..., g_m(x)]^T$
  $g(x) : \mathbb{R}^n \to \mathbb{R}^m$
- $\nabla g(x) = [\nabla g_1(x), \nabla g_2(x), ..., \nabla g_m(x)]$

We want to minimize the loss function defined by:

$$\boxed{f(x) = \frac{1}{2}\sum_{i=1}^{m}(\phi(d_i, x) - y_i)^2} \tag{3}$$

This loss function is also known as *Squared Loss Function.*
So using equation 3, our objective function becomes:

$$\min_{x \in \mathbb{R}^n} f(x) = \min_{x \in \mathbb{R}^n} \frac{1}{2}||g(x)||^2$$
$$\text{where } ||g(x)||^2 = \sum_{i=1}^{m} ||g_i(x)||^2$$

To find the best weights x of a neural network which reduces the squared error, we must keep updating our x in the direction of local minima of the objective function. Hence at k^{th} iteration the internal parameters will be x_k . Using first order Taylor series expansion we approximate g to $\tilde{g}$ at x_k as:

$$\underbrace{\tilde{g}(x, x_k)}_{m \times 1} = \underbrace{g(x_k)}_{m \times 1} + \underbrace{\nabla g(x_k)^T}_{m \times n}\underbrace{(x - x_k)}_{n \times 1}$$

This is also called linear approximation of $g(x)$ at point x_k.
So now, our objective function becomes:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}||\tilde{g}(x, x_k)||^2 \tag{4}$$

Expanding equation 4 using this approximation, we get:

$$\begin{aligned}
||\tilde{g}(x, x_k)|| &= \tilde{g}(x, x_k)^T \cdot \tilde{g}(x, x_k) \\
&= [g(x_k) + \nabla g(x, x_k)^T(x - x_k)]^T[g(x_k) + \nabla g(x, x_k)^T(x - x_k)]^T \\
&= \underbrace{||g(x_k)||^2}_{c^2} + 2\underbrace{(x - x_k)^T}_{x}\underbrace{\nabla g(x_k) g(x_k)}_{b} + (x - x_k)^T \underbrace{\nabla g(x_k)\nabla g(x_k)^T}_{a^2}(x - x_k)
\end{aligned}$$

So to get the minimum, we take the derivative of the above equation and equate it to 0.

$$f(x) = c^2 + 2xb + a^2x^2$$
$$\frac{df(x)}{dx} = 2b + 2a^2x$$

Equating it to 0, we get:

$$x = (-a^2)^{-1}b$$

Similarly, we can rewrite as:

$$\underbrace{x - x_k}_{n \times 1} = -(\underbrace{\nabla g(x_k)}_{n \times m}\underbrace{\nabla g(x_k)^T}_{m \times n})^{-1} \cdot \underbrace{\nabla g(x_k)}_{n \times m}\underbrace{g(x_k)}_{m \times 1} \tag{5}$$

Using equation 5, we can write this as:

$$\boxed{x_{k+1} = x_k - (\nabla g(x_k)\nabla g(x_k)^T)^{-1} \cdot \nabla g(x_k) g(x_k)}$$

Let's say,

$$D_k = (\nabla g(x_k)\nabla g(x_k)^T)^{-1}$$

We are assuming that D_k is invertible. If D_k is positive definite, then it is the same form as gradient descent algorithm. However, if it is not positive definite, we can make it by adding some positive values Δ_k to its diagonal elements. So now,

$$D_k = (\nabla g(x_k) \nabla g(x_k)^T + \Delta_k)^{-1}$$

So now our update is:

$$\boxed{x_{k+1} = x_k - \alpha_k D_k \cdot \nabla g(x_k) g(x_k)}$$

This method is known as *Gauss-Newton* method. This method was used before back-propagation method and SGD method to train the neural network.

Rank of the above D_k matrix is bounded by $min(m, n)$. We want rank to be at least n and to satisfy this condition we need $m >= n$. This says that number of data points must be greater than or equal to the number of parameters.
We have,

$$\begin{aligned}
\min_{x \in \mathbb{R}^n} f(x) &= \frac{1}{2} ||g(x)||^2 \\
&= \frac{1}{2} \left(\sum_{i=1}^{m} g_i(x)^2 \right) \\
\nabla f(x) &= \sum_{i=1}^{m} g_i(x) \nabla g_i(x) \\
\underbrace{\nabla f(x)}_{n \times 1} &= \underbrace{\nabla g(x)}_{n \times m} \underbrace{g(x)}_{m \times 1}
\end{aligned}$$

Hessian of $\nabla f(x)$ is,

$$\nabla^2 f(x) = (\nabla g(x) \nabla g(x)^T) + \sum_{i=1}^{m} g_i(x) \nabla^2 g_i(x)$$

This approximates Newton's method by ignoring the second term.

3 Need for a Different Search Method

3.1 Introduction

Till now we have seen various kinds of methods to minimize a function such as steepest descent, Newtons methods.These methods involve finding the minima via repeated iterations of the form:

$$x_{k+1} = x_k - \alpha_k D_k \nabla f(x_k)$$

where α_k is the step size which decides the rate at which we decrease our x_k value.

If the step size is fixed then it might converge on only very specific cases. Otherwise the value of $f(x_k)$ might diverge or oscillate. Hence, there is no guarantee of convergence to a local minima and we may run into problems. In order to avoid the above problems, instead of keeping α as a fixed constant, we can vary it at every step so as to ensure faster convergence to a local minima.

For finding an appropriate α at every iteration, we create a function,

$$\begin{aligned}
g(\alpha) &= f(x_k + \alpha d_k) \\
\alpha_k \in \operatorname*{argmin}_{\alpha > 0} g(\alpha) &= f(x_k + \alpha d_k)
\end{aligned}$$

and attempt to find an α which satisfies

$$g'(\alpha) = 0$$

This is done so as to get an α which minimizes $g(\alpha)$.

To find the minimum we could also solved $\nabla f(x) = 0$. This requires us to solve n equation and n unknowns. But we chose solving $g'(\alpha) = 0$ because it requires us to solve 1 equation and 1 unknown which is computationally very efficient.
However, for some functions, it is hard to solve $g'(\alpha) = 0$ analytically. Let's see an example.

Example

Let us try to find out appropriate α_k value for the function,

$$f(x, y) = x^3 + y^3 \sin y$$

$$\text{with initial point } x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\nabla f(x, y) = \begin{bmatrix} 3x^2 \\ 3y^2 \sin y + y^3 \cos y \end{bmatrix}, \qquad \nabla f(x_0) = \begin{bmatrix} 3 \\ 3.0647 \end{bmatrix}$$

$$x_1 = x_0 - \alpha \begin{bmatrix} 3 \\ 3.0647 \end{bmatrix}$$

$$\begin{aligned} g(\alpha) &= f(x_0 - \alpha \nabla f(x_0)) \\ &= f\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \alpha \begin{bmatrix} 3 \\ 3.0647 \end{bmatrix} \right) \\ &= f\left(\begin{bmatrix} 1 - 3\alpha \\ 1 - 3.0647\alpha \end{bmatrix} \right) \\ &= (1 - 3\alpha)^3 + (1 - 3.0647\alpha)^3 \sin(1 - 3.0647\alpha) \end{aligned}$$

Now to get an appropriate α, we need to solve $g'(\alpha) = 0$. However, we cannot analytically solve it very easily. We need to look for another method to find appropriate α.

3.2 Introduction to Line Search Method

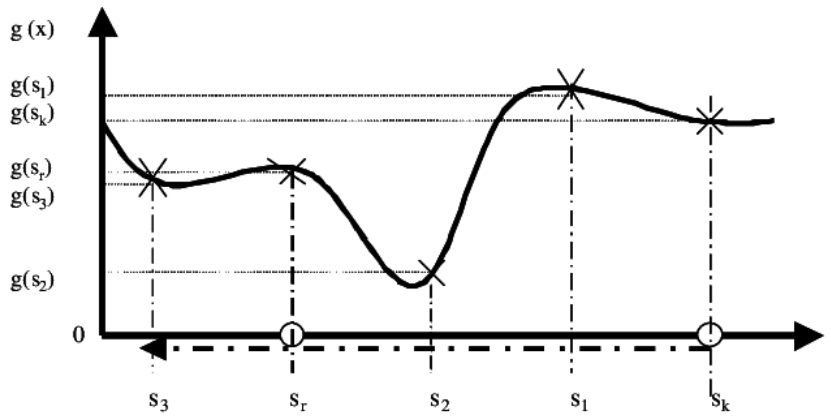

Figure 3: Graphical Interpretation of Line Search Method.[1]

The idea of a line search is to use the direction of the chosen step, but to control the length, by solving a one-dimensional problem of minimizing

$$g(\alpha) = f(x_k + \alpha p_k)$$

where p_k is search direction chosen from the position x_k.

$$g'(\alpha) = \nabla f(x_k + \alpha p_k) \cdot p_k$$

So, if we can compute the gradient, we can effectively do a one-dimensional search with derivatives.

References

[1] Line Search Method. https://www.researchgate.net/publication/2539573_Real-Coded_Evolutionary_Approaches_to_Unconstrained_Numerical_Optimization/figures?lo=1.