# Chapter 6

# Optimization: Review

In chapter 5, we have introduced the simplest machine learning algorithms, where the learned function is linear in the unknown weights. Even for these simple cases we can't find a closed form solution for logistic regression. What happens if we use more complicated function of the unknown weights. Of course, in most cases we won't be able to solve for the minima in closed forms. In this chapter we provide some background of the optimization techniques we will use to find the minima of non-linear loss functions for AI and ML applications.

**Question:** Is the span of fixed basis sufficient to capture complex natural phenomenons?

In real life applications, in most cases, we can not write the unknown function as a linear combination of fixed basis. In fact, we are mostly interested in learning the basis jointly with the weights in the form of a highly non-linear function.

So, can we solve for the optimum of these highly non-linear functions in closed form. The simple answer is, no, we can not. But, we have some iterative techniques to solve for the optimum that we implement for most of these learning approaches and which work very well.

## 6.1 Review: Finding the Stationary Points of a Non-linear Function

Let's assume that we have a function $g(x)$ the derivative of which $f(x) = g'(x)$ is non-linear in $x$ and we want to find stationary points of the function $g(x)$ what can we do?

One way to solve this problem will be to use Newton-Raphson method for finding the roots of $f(x)$. Newton-Raphson method is an iterative scheme where

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{6.1}$$

The above converges when $f(x) = 0$, notice that $f(x)$ corresponds to the roots of the gradient and will give us a stationary points of the function $g(x)$.

**Question:** Try to derive the expression above for Newton-Raphson method. Hint: Write the first-order Taylor series approximation of $f(x)$ and solve for $\Delta x$.

## 6.2 Gradient Descent

**Note:** Gradient is the direction of maximum change for a multi-variable function. For a function $f(x, y, z)$ gradient is defined as: $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}]^T$.

**Note:** Notice, that for we have defined gradient decent algorithm for a function of single variable but it can be easily extended to function of multiple variables.



Generally speaking, we are not interested in all types of stationary points. In optimization problems we are interested in either maxima or minima. The problem for finding maxima and minima are easily interchangeable by multiplying the function with negative 1. So, in general if have an algorithm for finding minima we can solve typical optimization problems.

The simplest algorithm in optimization for finding local minima is gradient decent. Gradient decent algorithm translates into taking a step locally in a direction where the function value is decreasing the most (i.e. negative gradient direction), and repeating the process until we are at a location of zero gradient (or very small gradient).

Let's assume that $f(x)$ is a function and we want to find a local minima for this function. Assume that we can calculate the function value and it's gradient and that we start with a point $x_0$, the gradient decent step will be

$$x_1 = x_0 - \mu f'(x_0) \tag{6.2}$$

where, $\mu$ controls the step size and $f'$ denotes the gradient of the function $f$.

We can repeat the above step until we reach a point where $f' = 0$ where the algorithm will converge. Notice the $f' = 0$ corresponds to the critical points of the function $f$ and since we are going in decent direction of function value (with very high probability) we will converge to a local minima.

**Algorithm 1:** Gradient Decent Algorithm

**Result:** Local minima
initialize $x_0$;
**while** $f'(x_n) > \epsilon$ **do**
  $x_{n+1} = x_n - \mu f'(x_n)$;
**end**

## 6.3 Gradient Decent with Backtracking:

In Algorithm 1 we provide simple gradient decent algorithm where step-size in each iteration is controlled by the hyperparameter $\mu$. Ideally speaking at each iteration we would the value of $\mu$ which takes us closest to a local minima. Will a bigger value of $\mu$ always takes us closer to a minima?

**Note:** Even in the simplest case of quadratic complex function, for a sufficiently large value of $\mu$ we can increase the value of $f(x)$ in algorithm 1. Unless of course we are at a minima where the algorithm will converge.



There are algorithm which ensures that we tune the value of $\mu$ at step so that we can ensure sufficient decrease of the function $f(x)$ at each iteration. One such algorithm is Backtracking summarized in Algorithm 2

**Algorithm 2:** Gradient Decent with Backtracking Algorithm

**Result:** Local minima
initialize $x_0$, $\mu = 1$ $\alpha = 0.1$ $\beta = 0.5$;
**while** $f'(x_n) > \epsilon$ **do**
  **while** $f(x_n) - \mu * f'(x_n) > f(x_n) - \alpha * f'(x_n)$ **do**
    $\mu = \mu * \beta$;
  **end**
  $x_{n+1} = x_n - \mu f'(x_n)$;
**end**

Notice, that in backtracking, we start with a large value of $\mu$ and decrease it until a sufficient decrease condition is met.

**Good Practice:** In both `PyTorch` and `TensorFlow` , we should use decaying learning rates for training networks.

## 6.4 Stochastic Gradient Decent

So far, we have seen methods which are dependent on calculation of the gradient of the loss function for finding local minima. Assuming that our loss functions are differentiable can we always find the gradient?

Notice, that the loss functions are summation over all samples of data. For deep learning application we have a huge number of training samples, running beyond millions in some cases. This makes the computation of gradient very expensive. A way around this issue is to use stochastic gradient decent where rather then computing the gradient of the loss function for the entire dataset we calculate the gradient for randomly selected batch of data in each iteration.

Let's assume that our loss function is defined as

$$L(\boldsymbol{W}) = \sum_{x_i \in D} l_{\boldsymbol{W}}(y_i, \hat{y}(x_i)) \tag{6.3}$$

where, $l$ is a loss function $(x_i, y_i)$ is a training sample and $D$ is the training data set. The stochastic gradient decent algorithm is provide below:

**Algorithm 3:** Stochastic Gradient Decent Algorithm

**Result:** Local minima
initialize $\boldsymbol{W}_0$;
**while** $\underline{||\boldsymbol{W}_{i+1} - \boldsymbol{W}_i||_2^2 > \epsilon}$ **do**
  $B \leftarrow$ random subset of D;
  $L(\boldsymbol{W}) = \sum_{x_j \in B} l_{\boldsymbol{W}}(y_j, \hat{y}(x_j))$;
  $\boldsymbol{W}_{n+1} = \boldsymbol{W}_n - \mu \nabla_{\boldsymbol{W}} L(\boldsymbol{W})$;
**end**

Stochastic gradient decent is the bread and butter of machine learning and artificial intelligence algorithm. Familiarize yourself well with this algorithm as you will be using it frequently.

## 6.5 Second Order Methods: Newton Method

*This section is just for providing an overview and can be skipped* All of the techniques we discussed above work with the first order Taylor series approximation of the functions. We can also use higher order approximation of the function for setting up our iterative schemes. The most common second order technique is Newton method. The algorithm for Newton method is summarized in Algorithm 4

**Algorithm 4:** Newton Method Algorithm

**Result:** Local minima
initialize $x_0$;
**while** $\underline{f'(x_n) > \epsilon}$ **do**
  $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$;
**end**

**Note:** Newton method converges in fewer iterations to the local minima.

**Question:** Is there any relation between Newton-Raphson method for root finding and Newton method for finding minima?

**Question:** Does the function value decrease in each iteration of the Newton method?

**Question:** Can you prove using the second order Taylor series that the Newton method step is the best step we can take?

### Exercises

**Exercise 6.1:** Write a python script to perform gradient decent on a function. Also implement the Newton method. Compare the number of iterations of convergence in both cases.