Laboratory Assessment: Machine Learning (COMS 3007) (Lab Session 2)

Devon Jarvis: 1365149

February 26, 2018

1 Optimization: Direct Approach

1. 1) For any given data set, model and objective function which is differentiable in its parameters it is possible to find the optimal values for the parameters using partial derivatives. To obtain these values one must take the first order partial derivative of the objective function with respect to all the parameters and set these partial derivatives equal to 0. These expressions of the partial derivatives can then be solved simultaneously to obtain the local extrema in the data (where all partial derivatives are equal to 0). To be a good objective function, it will have to have a local minima. As we often choose the objective function, we can ensure this is the case by design and thus the local extrema found from this method will have at least one local minima. The values of the parameters at this local minima will be the optimal values of the parameters given the data set, model and objective function.

2) a)

$$
\begin{aligned}
E(\theta) &= \Sigma i(\hat{y}i - yi)^2 \\
&= \Sigma i(\theta_1 xi + \theta_2 - \hat{y}i+)^2 \\
&= -\Sigma i(\hat{y}i - \theta_1 xi - \theta_2)^2 \\
\frac{\partial \Sigma i}{\partial \theta_2} &= -2\Sigma(yi - \theta_1 xi - \theta_2)(-1) \\
\frac{\partial \Sigma i}{\partial \theta_2} &= 2\Sigma(yi - \theta_1 xi - \theta_2) \\
Let: & \\
\Sigma yi - \theta_1 \Sigma xi - n\theta_2 &= 0 \\
\Rightarrow \Sigma yi &= \theta_1 \Sigma xi + n\theta_2 \\
\Rightarrow n\theta_2 &= \Sigma yi - \theta_1 \Sigma xi \\
\Rightarrow \theta_2 &= \frac{1}{n}\Sigma yi - \frac{\theta_1}{n}\Sigma xi \\
\Rightarrow \theta_2 &= \bar{y} - \theta_1 \bar{x}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \Sigma i}{\partial \theta_1} &= -2\Sigma(yi - \theta_1 xi - \theta_2)(-xi) \\
\frac{\partial \Sigma i}{\partial \theta_2} &= 2\Sigma(xiyi - \theta_1 xi^2 - \theta_2 xi) \\
Let & \\
\Sigma xiyi - \theta_1 \Sigma xi^2 - \theta_2 \Sigma xi &= 0 \\
\Rightarrow \Sigma xiyi &= \theta_1 \Sigma xi^2 + \theta_2 \Sigma xi \\
\Rightarrow \Sigma xiyi &= \theta_1 \Sigma xi^2 + (\frac{1}{n}\Sigma yi - \frac{\theta_1}{n}\Sigma xi)\Sigma xi \\
\Rightarrow \Sigma xiyi &= \theta_1 \Sigma xi^2 + (\frac{1}{n}\Sigma xiyi - \frac{\theta_1}{n}\Sigma xi^2) \\
\Rightarrow \Sigma xiyi &= \theta_1(\Sigma xi^2 - \frac{1}{n}\Sigma xi^2) + \frac{1}{n}\Sigma xiyi \\
\Rightarrow \theta_1 &= \frac{\Sigma xiyi - \frac{1}{n}\Sigma xiyi}{\Sigma xi^2 - \frac{1}{n}\Sigma xi^2} \\
\Rightarrow \theta_1 &= \frac{\Sigma xiyi(1 - \frac{1}{n})}{\Sigma xi^2(1 - \frac{1}{n})} \\
\Rightarrow \theta_1 &= \frac{\Sigma xiyi}{\Sigma xi^2}
\end{aligned}
$$

b)

$$\hat{y}i = \theta_1 e^(\theta_2 xi)$$
$$ln(\hat{y}i) = ln(\theta_1) + ln(e^(\theta_2 xi))$$
$$ln(\hat{y}i) = ln(\theta_1) + \theta_2 xi$$
$$Let:$$
$$zi = ln(yi),$$
$$A_0 = ln(\theta_1)$$
$$and A_1 = \theta_2$$
$$zi = A_0 + A_1 xi$$

From the linear regression model defined in question 2a:

$$A_1 = \frac{\Sigma xizi - \frac{1}{n}\Sigma xizi}{\Sigma xi^2 - \frac{1}{n}\Sigma xi^2}$$
$$A_0 = \bar{z} - \theta_1 \bar{x}$$
$$Finally$$
$$\theta_2 = A_1$$
$$\theta_1 = e^(A_1)$$
$$\Rightarrow \theta_1 = exp(\frac{\Sigma xizi - \frac{1}{n}\Sigma xizi}{\Sigma xi^2 - \frac{1}{n}\Sigma xi^2})$$

3) a) The grouping of the data points changes in response to the noise model. For example if a Gaussian distribution is used to generate noise, then there will be more data evenly spread around the state space for the parameters, with the large majoritory of data falling in the middle of the state space. If an exponential distribution is used on the other hand, then most of the data points will fall toward the lower end of the state space and the data will have lower values. Thus changing the functional form of the noise changes the distribution of the data.

b) Changing the parameter values of the noise model will change the shape of the model. The parameters determine the relationship between the input dimensions and output dimension of the model. Thus by changing the shape of the model different relationships between the dimensions and what they represent can be modelled or shown. For example in linear modelling/linear regression, one parameter represents the gradient of the output dimension with regard to the one of the input dimensions. Thus modeling the rate that the output values change in response to the input dimensions.

2 Optimization: Iterative Approach

1.

2. 1) Generally iterative optimization techniques need to be used for optimizing problems in higher dimensional domains where computing the partial derivatives for ever dimension and finding the solution directly from simultaneous equations becomes computationally expensive and difficult.

   a) Gradients provide a useful approach in an iterative sense as they indicate the best direction to move in to minimize the objective function. Further finding just the first order partial derivative is not very computationally expensive and can be performed frequently as a result, making the use of gradients logical for an ierative algorithm. Gradients are also continuous and as a result in many cases it is not necessary to re-compute the gradient at every time step and instead the gradients from the previous timestep can merely be updated to estimate the gradients at the current time step. This is a further efficiency boost and results in rapid estimations and updates of gradients.

   b) Another iterative approach that could be used is to randomly pick values in the state space and record the minimum of all the random choices. This is very inefficient, however can be performed very quickly and thus many guesses can be made in a short space of time. With enough guesses a good approximation of the local minimum of the objective function can be found, however there is no upper bound to how many choices it will take to get a "good" result and it is very unlikely that the best choices will be the actual minimum point of th objective function and will likely always remain a mere approxmation.

   2) **Netwon Raphson Method Univariant Derivation:**
   Given the function f(x), the derivative of f(x) at point a is:
   $f\prime(a) \approx \frac{f(x)-f(a)}{x-a}$
   $\Rightarrow x - a \approx -\frac{f(a)}{f\prime(a)}$
   as f(x) is 0 as Newton-Raphson method find roots of a function
   $\Rightarrow x \approx a - \frac{f(a)}{f\prime(a)}$

   a) This algorithm is useful in a machine learning context as it is a simple iterative algorithm for finding roots of a function. Thus it can be used as a simply algorithm to compute the root points of the function representing the gradients of an objective function,

and thus as a result can be used to find minima in the objective function. One draw-back to this algorithm us the use of the second order partial derivative, which is computationally expensive to compute and as a result as the number of dimensions increases, the time taken to perform the algorithm will scale exponentially. However in a reasonably small amount of dimensions this algorithm is ideal due to its simplicity and the fact that it converges quickly to minima points with relatively few iterations required before it converges in comparison to other first order algorithms such as gradient descent.

b) In a machine learning context, x_i are the parameters of the model, $f(x_i)$ is the function representing the derivatives of the objective function at each point on its domain, and $f'(x_i)$ is the second order derivative function of the objective function. The use of the first and second partial derivatives instead of the objective function and the first order partial derivative is due to the fact that Newton-Raphson finds roots of a function and not extrema. Thus the roots of the derivative function of the objective function must be found in order to find the points on the domain where the objective function is at a minima.

3) The uni-variant Newton-Raphson Method derived above can be generalzed to the multi-variable case to solve n simultaneous algebraic equations:

$f_1(x_1, ..., x_n) = f_1(x) = 0$
...
$f_n(x_1, ..., x_n) = f_n(x) = 0$

where $x = [x_1, ..., x_n]^T$ is an n-dimensional vector.
This equation system can be more concisely represented in vector form as $f(\mathbf{x}) = 0$. The multi-variate Newton-Raphson formula is

$x \Leftarrow x - J_f^-1(x)f(x)$

where $J_f(\mathbf{x})$ is the Jacobian of the function $f(\mathbf{x})$:
$J_f^-1 = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & ... & \frac{\partial f_1}{\partial x_n} \\ ... & ... & ... \\ \frac{\partial f_n}{\partial x_1} & ... & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$
To derive this we consider the Taylor series:
$f_i(\mathbf{x} + \delta\mathbf{x}) = f_i(\mathbf{x}) + \Sigma \frac{\partial f_i}{\partial x_j} \delta x_j + \mathcal{O}(\delta\mathbf{x}^2)$ *for* $(i = 1, ..., n)$
We ignore the term of $\delta\mathbf{x}^2$ and higher and let $f_i(\mathbf{x} + \delta\mathbf{x})$ be zero (ie: $\mathbf{x} + \delta\mathbf{x}$ is the zero-crossing of the tangent), and get:
$\Sigma_j \frac{\partial f_i}{\partial x_j} \delta x_j = -f_i(\mathbf{x})$ *for* $(i = 1, ..., n)$

Solving this linear equation system for δx_j, we get
$\delta \mathbf{x} = -J_f^-1(\mathbf{x})f(\mathbf{x})$
and the Newton-Raphson formula:
$\mathbf{x} \Leftarrow \mathbf{x} + \delta \mathbf{x} = \mathbf{x} - J_f^-1(\mathbf{x})f(\mathbf{x})$

4) Attached

5) For the experiment I aimed to see how the number of iterations needed for the algorithm to converge scaled in relation to A the number of input dimensions in the function, seen in Table 1 and B the highest power of the first dimension, seen in Table 2.
To remain consistant restrictions were made in both experiments. For A the algorithm started at coordinate 1,75 for every dimension always, and the only dimension with a power was the first one which was to the power of 2 (to increase difficulty of convergence and thus make it easier to discern how convergence rate changed.) For B the function remained with 2 input dimensions and only the power of the first dimension was changed. All coordinates were again started at 1,75.
From 1 it can be seen that the number of input dimensions has no real effect on convergence rate. The relationship seems to model a logerithmic curve
From 2 it can be seen that there is a relatively linear relationship between convergence rate and the highest power in the function.
Note that the number of interations expressed in the results is an average taken over 5 repetitions of the algorithm.

Table 1: Experiment A

Num Input Dimensions	**Average Num Iterations**
α	β
2	4
3	6
4	6
5	6

6) a) The addition of a learning rate could be added. This would increase the change in the parameters by a larger amount per iteration and thus could cause faster learning and thus convergence. A potential problem with this is that it increases the likelyhood of the algorithm oscillating around a minimum point however not converging as it can't make specific or minimal enough changes to the parameters to find the minima. Thus the function would "over-shoot" the parameter values required to converge every iteration.

Table 2: Experiment B

Highest power γ	**Average Num Iterations** β
1	2
2	4
3	7
4	5
5	9

b) Topologically, very flat curves (and as a result very elastic or volatile parameters) would struggle to converge as the model would make very large changes in the parameter values as a result. Furthermore any discontinuities in the input dimensions would make it impossible for the algorithm to explore the entirity of the solution space and as a result may be unable to find minima in the function. This would occur in the case of a donut like function for example. Finally, bounded soultion spaces may also not converge as the gradients found may lead the model to move towards having parameters that result in an undefined output (an output outside of the range of possible value). Restrictions that could be imposed would be to restrict the parameters to within a certain range of the center of the model, or if the center of the model is hard to determine, the 0,0. Naturally the parameters would have to be restricted from falling into discontinuities, special exceptions in the algorithm will need to be made to compensate for the existence of these discontinuities.

c) Geometrically smooth curves will result in better performance from the algorithm than curves that are as smooth, or wrinkly. This is due to the fact that smooth curves have much more consistant gradients and thus the parameters will be updated much more consistently over the iterations. Non-smooth curves on the other hand have very variable gradients and as a result the parameter values might be updated in a way which does not optimize or aid in optimization for some iterations. Thus training and convergence would take much longer as the algorithm would have to overcome some of these negative updates and its good updates in general would be less beneficial to the network, again due to the variability in the gradients. In extreme cases wrinkly curves may stop the algorithm from finding a minimum at all, as the algorithm may get caught in a "wrinkly" which it would perceive to me a minimum value of the function, when in fact it is not