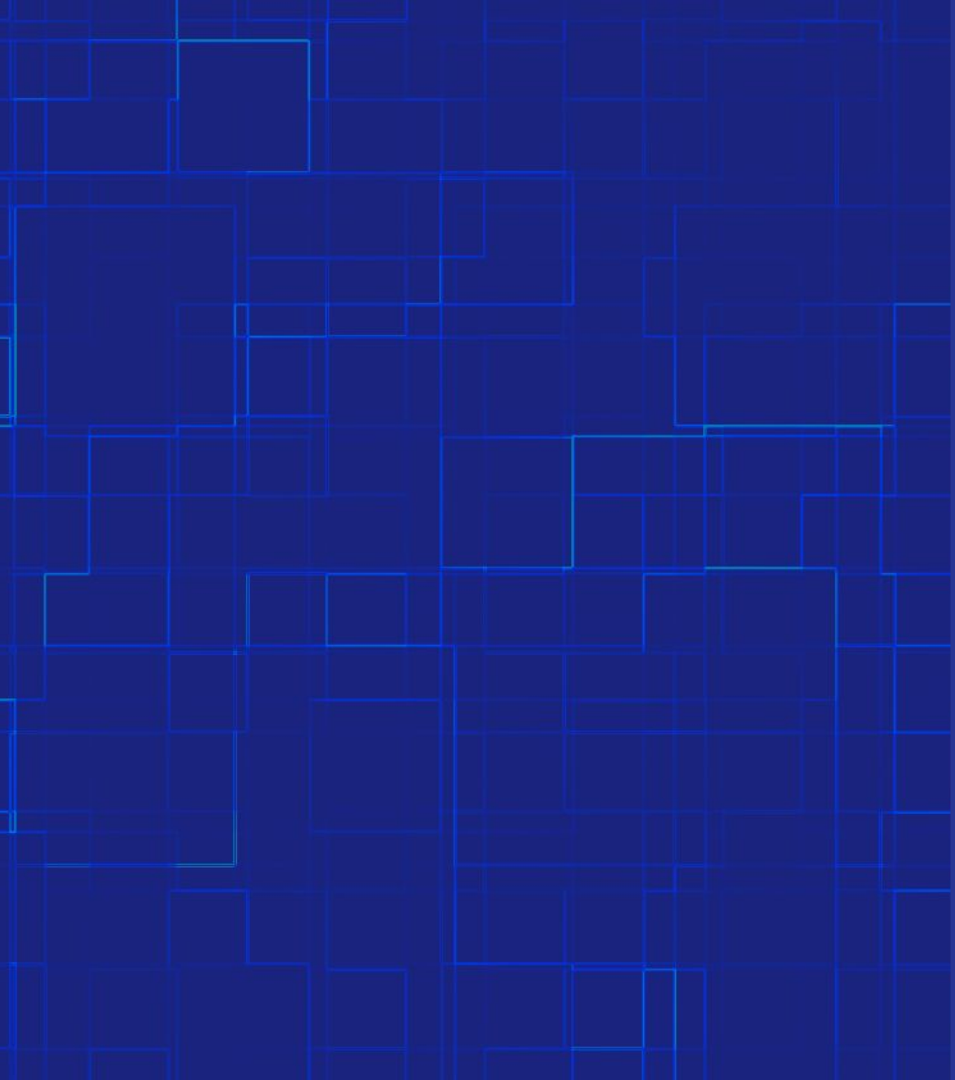


# Second order methods in Training Neural Networks

Avani Gupta, 2019121004

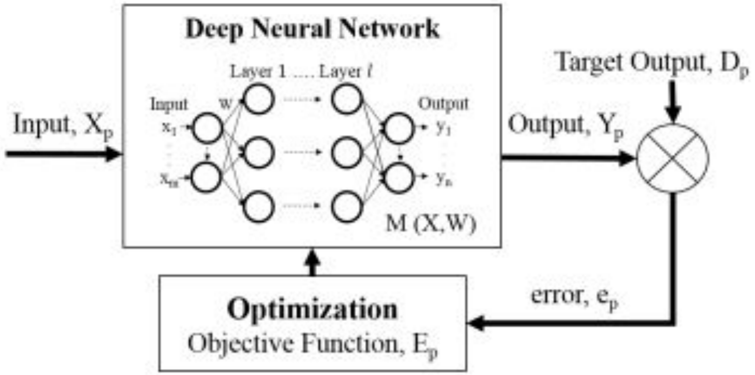
## Training Neural Networks

- Neural Networks are used as function approximators which try to learn and predict the distribution of data.
- They have neurons with activation functions which work in unison.
- A neuron basically is a mathematical function which takes inputs $x_1$, $x_2$, $x_{\square}$. Each input has a corresponding weight value

  $w_1$, $w_2$, $w_k$. The output of neuron is given as

  $f(x_1, x_2, x_n) = w_1x_1 + w_2x_2 + w_nx_n$ where $x_i$ are the inputs to

  neuron.
- During training, the loss is calculated and weights are updated via back-propagated.
- During testing the weights are fixed.



# Optimization in Neural Networks

- Backpropagation is a step in optimization techniques that measures error from the targeted and calculated performance.
- The aim of Optimization is to minimize the target function, $E_p$, against the solution, W* for minimum error value. The equation for iterative updates is as follows:

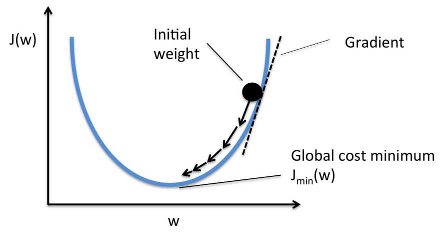
$$W_{k+1} = W_k - \eta \frac{\partial E}{\partial W}$$

where $W_{k+1}$ is the updated weight matrix, k is the iteration, η

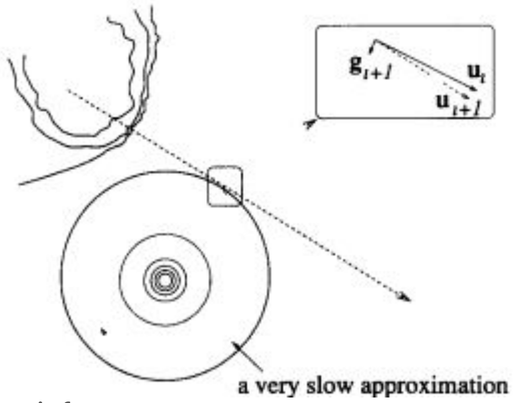
is the step length.

The gradient descent approach is one of the most widely used optimization methods in artificial neural network training. It uses first-order derivatives. The negative gradient is used to move across the error surfaces of gradient descent.

$$-\frac{\partial E}{\partial W} = -\nabla E$$



# First order vs Second Order methods



## First order

Use gradient information to construct the next training iteration

First order methods requires heavy fine-tuning of hyper-parameters since they rely on just the gradient information.

## Second order

Use Hessian to compute the iteration based on the optimization trajectory

The second order methods are able to take into account the curvature and hence observe better performance of model.

# Second order methods for training of neural networks

## Newton's method

- It uses inverse Hessian matrices to compute the next update step.

$$W_{k+1} = W_k - \eta(\nabla E_k + \beta_k p_{k-1})$$

- Time complexity $O(N^3)$ per iteration, space complexity of order $O(N^2)$.

## Conjugate gradient method

- It is an Approximate Hessian Method.
- It computes the conjugate direction and adaptively alters the direction of descent in each iteration. Conjugate direction is computed as: $\beta_k = \frac{(\nabla E_k - \nabla E_{k-1})^T \nabla E_k}{(\nabla E_{k-1})^T \nabla E_{k-1}}$
- Update rule: $W_{k+1} = W_k - \eta(\nabla E_k + \beta_k p_{k-1})$
- Time complexity: O(N )

# Second order methods for training of neural networks

## Quasi-Newton Method

It computes updates using inverse Hessian estimation. The Broyden-Fletcher-Golfarb-Shanno (BFGS) algorithm is used for approximation.

Update equation: $W_{k+1} = W_k - \eta Q_k \frac{\partial E}{\partial w}$

Time complexity: O(N $^2$) ; Space Complexity: O(N $^2$)

## Gauss Newton

Gauss-Newton method approximates inverse Hessian using the square of Jacobian(truncated Hessian) ignoring the second-terms.
Update equation:

$$W_{k+1} = W_k - \eta \left( \frac{\partial E^T}{\partial W} \frac{\partial E}{\partial W} + \mu I \right)^{-1} \frac{\partial E}{\partial W}$$

Time complexity: O(N $^2$) ; Space Complexity: None

## Second order methods for training of neural networks

### Levenberg Marquardt

To tackle the problem of limitless Hessian, the Levenberg-Marquardt (LM) uses a regularisation parameter, with η > 0. Levenberg-Marquardt method is written as follows:

$$W_{k+1} = W_k - \eta \left( \frac{\partial E^T}{\partial W} \frac{\partial E}{\partial W} + \mu I \right)^{-1} \frac{\partial E}{\partial W}$$

### Approximate Greatest Descent method

AGD uses the trust area approach to find the smallest points depending on a set of boundaries.

$$W_{k+1} = W_k - \eta (H + \mu I)^{-1} \frac{\partial E}{\partial W}) where (\mu = ||\nabla E|| / r$$

## Second order methods for training of neural networks

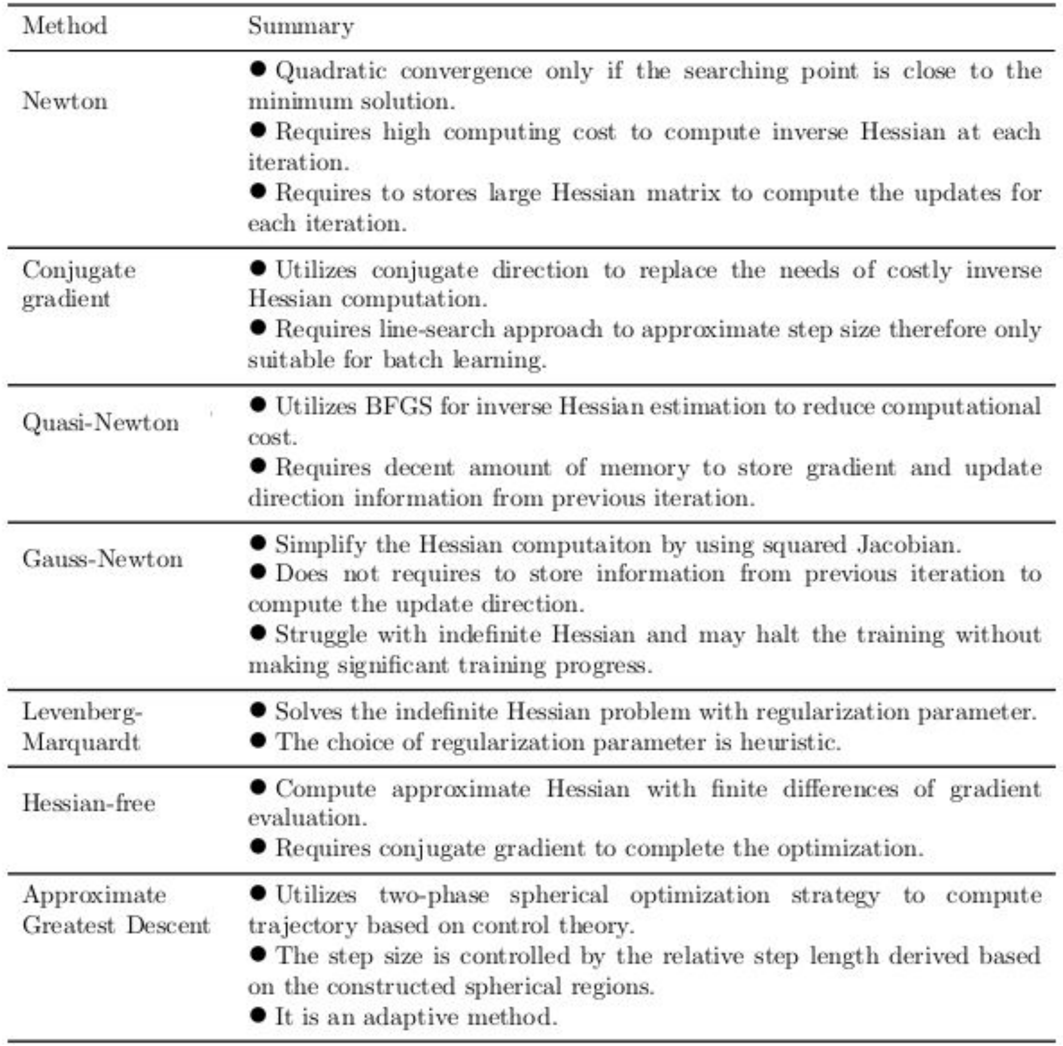
### Hessian-free method

It's also known as truncated-Newton because it uses a scaled-down version of Hessian to calculate the local curvature before applying conjugate gradient to further optimise

Update equation:

$$W_{k+1} = W_k - \eta Q_k \frac{\partial E}{\partial w}$$

## Summary of second-order methods



| Method | Summary |
|---|---|
| Newton | ● Quadratic convergence only if the searching point is close to the minimum solution. ● Requires high computing cost to compute inverse Hessian at each iteration. ● Requires to stores large Hessian matrix to compute the updates for each iteration. |
| Conjugate gradient | ● Utilizes conjugate direction to replace the needs of costly inverse Hessian computation. ● Requires line-search approach to approximate step size therefore only suitable for batch learning. |
| Quasi-Newton | ● Utilizes BFGS for inverse Hessian estimation to reduce computational cost. ● Requires decent amount of memory to store gradient and update direction information from previous iteration. |
| Gauss-Newton | ● Simplify the Hessian computaiton by using squared Jacobian. ● Does not requires to store information from previous iteration to compute the update direction. ● Struggle with indefinite Hessian and may halt the training without making significant training progress. |
| Levenberg-Marquardt | ● Solves the indefinite Hessian problem with regularization parameter. ● The choice of regularization parameter is heuristic. |
| Hessian-free | ● Compute approximate Hessian with finite differences of gradient evaluation. ● Requires conjugate gradient to complete the optimization. |
| Approximate Greatest Descent | ● Utilizes two-phase spherical optimization strategy to compute trajectory based on control theory. ● The step size is controlled by the relative step length derived based on the constructed spherical regions. ● It is an adaptive method. |

## Experimental Settings

**Goal: Estimate the Gaussian given initial estimate.**

**Learning rate** 0.01 for Gradient Descent and Gauss Newton, 10 for LM (experimentally found:see graphs above)

**Tolerance** $1e^{-2}$ for Gradient Descent, Gauss Newton and LM

**Damping factor for LM** 1.1 (note: this is intentionally kept 1.1 since if decrease in loss learning rate should decrease so that we are more probable to not miss the local optima. And for increase in loss it is intended to increase learning rate since algo is going farther away from optima(or shooting it). Even the authors of LM algo(http://people.duke.edu/~hpgavin/ce281/lm.pdf) suggested to use damping factor greater than 1

- The Gradient Descent, Gauss Newton and LM converge for above values of learning rate and tolerance in given estimate of $s_{gt} = 20$, $a_{est} = 10$, $m_{est} = 13$, $s_{est} = 19.12$ and 50 observations.

Code: https://colab.research.google.com/drive/1sHeR5j1R_gd8vVBN_Mm51R201fOVdWLx?usp=sharing

## Experimentation Results

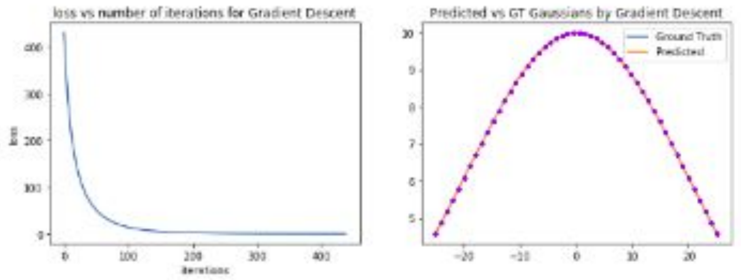


Fig. 1: Gradient Descent performance

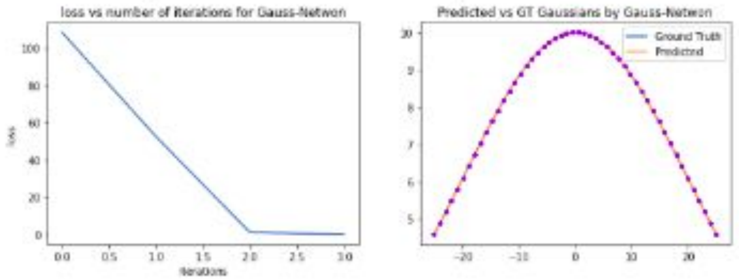


Fig. 2: Gauss Newton performance

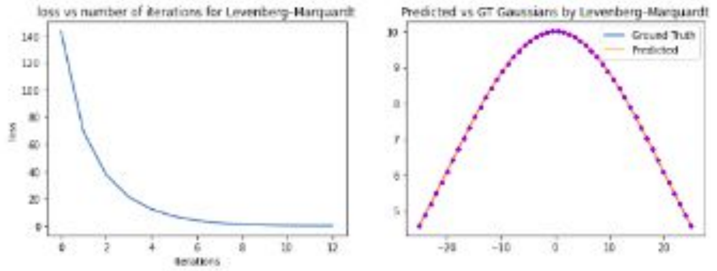
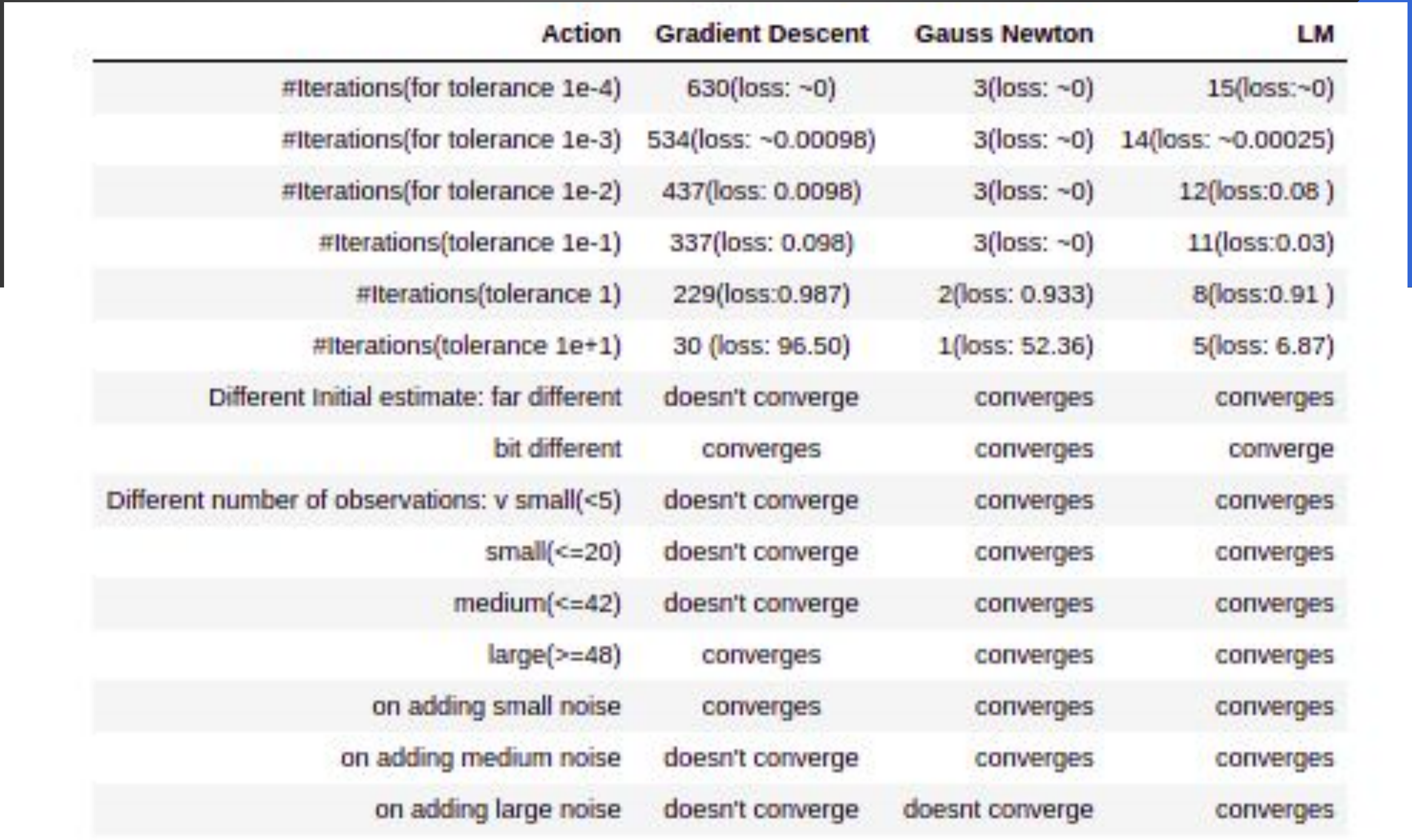


Fig. 3: LM performance

## Experimental obs:



| Action | Gradient Descent | Gauss Newton | LM |
|---|---|---|---|
| #Iterations(for tolerance 1e-4) | 630(loss: ~0) | 3(loss: ~0) | 15(loss:~0) |
| #Iterations(for tolerance 1e-3) | 534(loss: ~0.00098) | 3(loss: ~0) | 14(loss: ~0.00025) |
| #Iterations(for tolerance 1e-2) | 437(loss: 0.0098) | 3(loss: ~0) | 12(loss:0.08 ) |
| #Iterations(tolerance 1e-1) | 337(loss: 0.098) | 3(loss: ~0) | 11(loss:0.03) |
| #Iterations(tolerance 1) | 229(loss:0.987) | 2(loss: 0.933) | 8(loss:0.91 ) |
| #Iterations(tolerance 1e+1) | 30 (loss: 96.50) | 1(loss: 52.36) | 5(loss: 6.87) |
| Different Initial estimate: far different | doesn't converge | converges | converges |
| bit different | converges | converges | converge |
| Different number of observations: v small(<5) | doesn't converge | converges | converges |
| small(<=20) | doesn't converge | converges | converges |
| medium(<=42) | doesn't converge | converges | converges |
| large(>=48) | converges | converges | converges |
| on adding small noise | converges | converges | converges |
| on adding medium noise | doesn't converge | converges | converges |
| on adding large noise | doesn't converge | doesnt converge | converges |

https://colab.research.google.com/drive/1sHeR5j1R_gd8vVBN_Mm51R201fOVdWLx?usp=sharing

## Observations

- the Gauss-Newton(GN) method and Levenberg-Marquardt(LM) typically converge much faster than gradient-descent(GD) methods. This is attributed to fact that former take larger steps than GD.
- Larger steps are desired if loss is increased since the algos prediction is far from optima and smaller steps are desired when loss is decreased attributing to fact that algo is near optima(so that it doesn't overshoot the optima). This is achieved by Levenberg-Marquardt algorithm by changing learning rate dynamically according to loss incurred. Thus LM takes larger steps when far away from optima and smaller steps when it is close to optima and hence converges better. Note loss is decreased by LM but since it starts taking smaller steps when near optima iterations may increase to converge(as compared to GN)

**Different initial estimates**

- The Levenberg-Marquardt method acts more like a gradient-descent method when the parameters are far from their optimal value, and acts more like the Gauss-Newton method when the parameters are close to their optimal value.
- On varying initial estimates, number of observations and addition of noise LM and Gauss newton perform well. This is attributed to fact that those algorithms take large steps and thus are able to converge for far off initial estimates too.

# Observations

**Smaller number of observations**

- Gradient descent is not able to converge on small number of observations since gradient steps are much smaller based on learning rate, where as the steps in GN and LM are much larger.

**Noisy Objective function**

- For higher levels of noise LM is still able to handle better than GD or GN. For medium noise only LM and Gauss Newton converge, for low noise all converge, given learning rate and tolerance kept fixed. This is due to fact that LM and GN takes larger steps and hence it does'nt get stuck at optima which is incurred due to noise in GD.

- Thus to conclude our order of superiority of algorithms (from most cases above) is LM > GN > GD
- This is intuitive too since Levenberg-Marquardt theoretically combines pros of Gauss Newton and Gradient Descent.

## Conclusion

- The Newton's method was the first second-order optimization method proposed for neural networks training. It uses full Hessian in training and is prone to computation and memory issues.
- Quasi-Newton and Gauss-Newton were introduced to counter the drawback of Newton method with truncated Hessian and approximate Hessian respectively.
- Levenberg-Marquardt method was proposed to solve the indefinite matrix problem with the introduction of regularization parameter to the truncated Hessian's.
- Consequently, approximate greatest descent utilizes control theory in developing a two phase long term and short term optimization approach.
- The Hessian-free method was then proposed as an alternative second-order optimization technique with the help of conjugate gradient algorithm.
- Conjugate gradient and quasi-Newton both require to store information from previous iteration. Alternatively, Hessian-free method on Hessian approximation does not work as efficient without conjugate gradient algorithm.
- Although Gauss-Newton and Levenberg-Marquardt method requires more computation power per iterations, it can be solved with the help of parallel processing. Levenberg-Marquardt also uses less memory compared to other Hessian approximation approaches which enables the algorithm to run on lower-ends graphical processing unit.
- Approximate greatest descent utilizes an adaptive two phase methods that which had proven to provide better trajectory towards the minimum.