ANN1

Irene Volpe

r0740784

Exercise 2: Backpropagation in feedforward MLP

Function approximation: comparison of various algorithms

We compare performance of GD with:
a). GD adaptive learning rate (GDA)
b). Levenberg-Marquardt algorithm (LM)
c). BFGS quasi Newton algorithm (BFG)
d). Fletcher-Reeves conjugate gradient algorithm (CGF)
e). Polak-Ribiere conjugate gradient algorithm (CGP).
Two Feed-Forward Networks are created; first one is trained on GD and other on the algorithm to be compared.
Number of hidden layer neurons and values of bias and weights are kept same. Training is run for 1, 15 and 1000 epochs.

The algorithms are compared on noisy and noiseless data for; Speed, Mean Square Error, Regression and Gradient.
Generalization and overfitting are determined by Mean Squared Error (MSE).
Regression value shows correlation between outputs and targets; '1' means a close and '0' a random relationship.
The gradient 'G' becomes very small when training reaches minimum of performance.

To see the effects of generalization, two sets of experiments are run in batch mode:
1) the first on simple nonlinear function y = sin(x^2) for x = 0: 0:05: 3π
2) the second by adding Gaussian noise to our dataset.
The mean of speed and MSE are taken as the final results.

1 speed

Figure 1:
GD takes ≈1 second to train on both noisy and noiseless data. GDA, LM and CGP interestingly, take lesser time to train on noisy data. CGF takes much more time to train on noisy data. BFG has the highest computation time of 7 seconds in both cases.

2 MSE (performance)

Figure 2:
GD has highest MSE of more than 0.3 on noiseless data, which is slightly less (≈0.25) on noisy data. This is because GD generalizes well on given dataset. GDA has 2nd highest MSE of ≈0.15 on noiseless data and slightly higher on noisy data. All other algorithms perform better with MSE scores ≈zero on noisy and noiseless data.

3 Gradient and Regression

Figure 3:
1) GD algorithm has the lowest correlation value between output and target
2) All other algorithms have equally higher correlation values;
3) When training reaches minimum of performance, GD and GDA have the largest values for the gradients;
4) The gradients are generally bigger when the trainings stop if Gaussian noise is added to the dataset, especially if we're deploying GDA optimization algorithm.

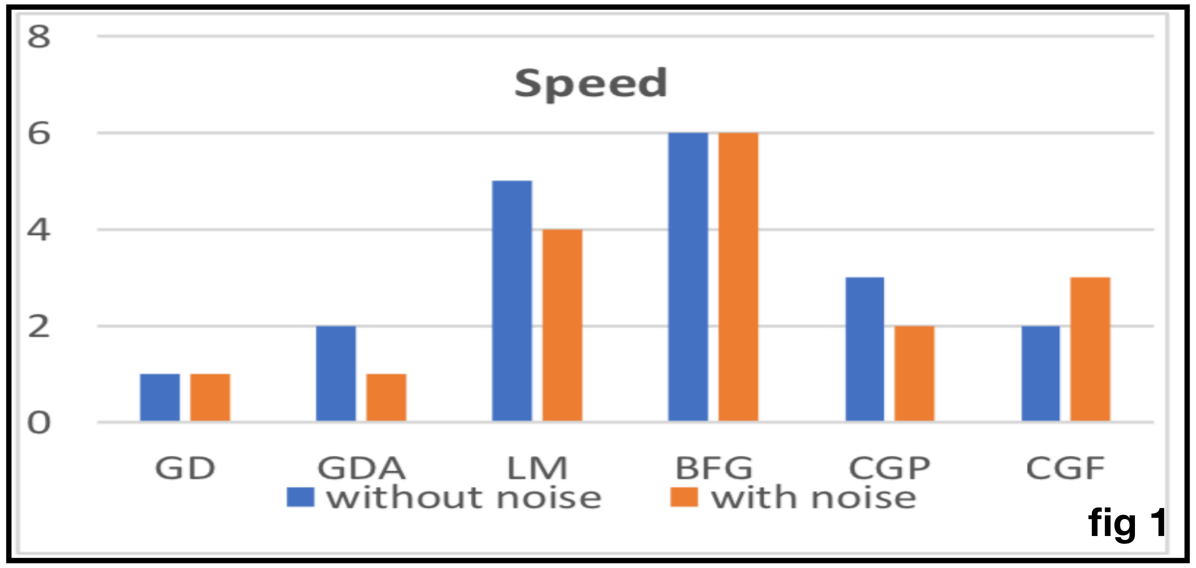


fig 1

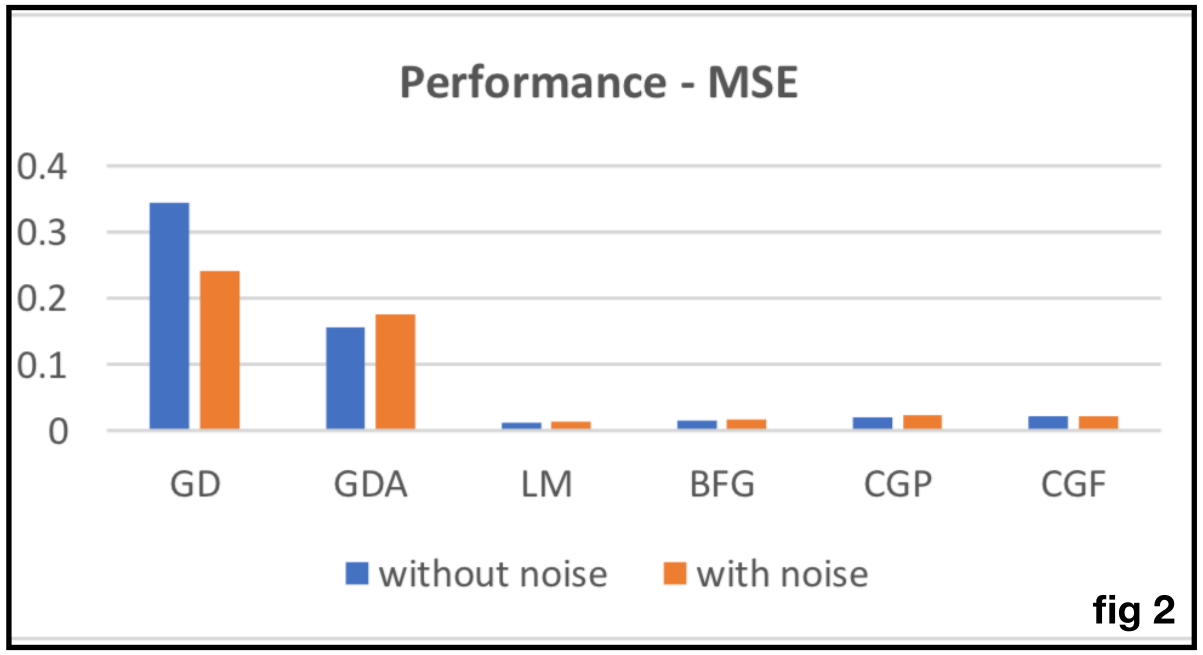
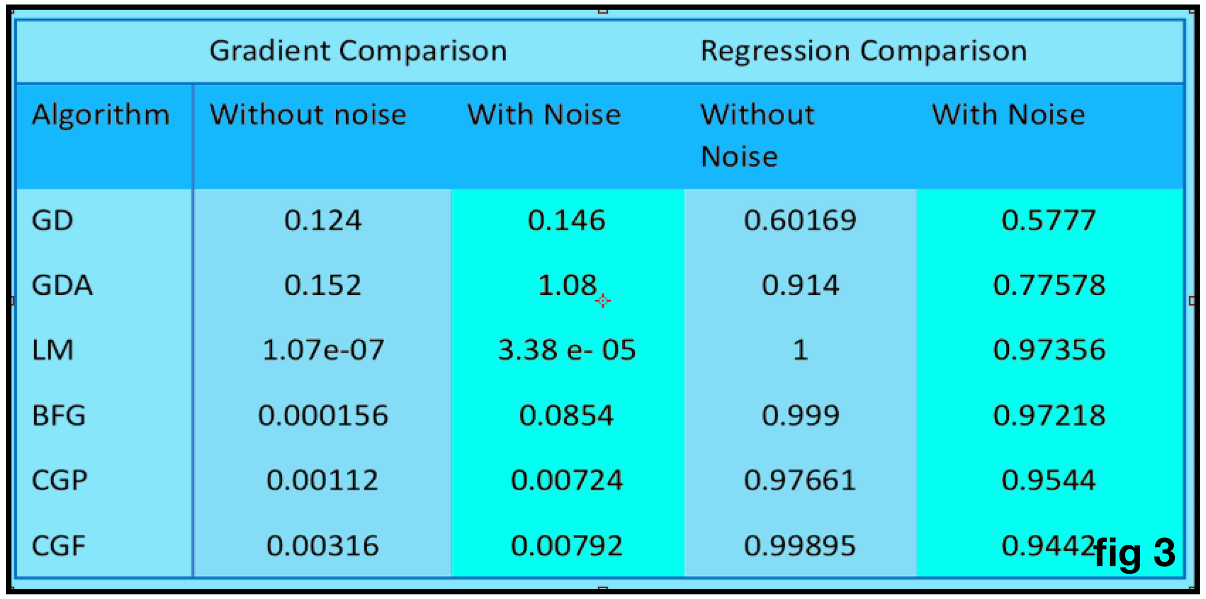


fig 2



	Gradient Comparison		Regression Comparison	
Algorithm	Without noise	With Noise	Without Noise	With Noise
GD	0.124	0.146	0.60169	0.5777
GDA	0.152	1.08	0.914	0.77578
LM	1.07e-07	3.38 e- 05	1	0.97356
BFG	0.000156	0.0854	0.999	0.97218
CGP	0.00112	0.00724	0.97661	0.9544
CGF	0.00316	0.00792	0.99895	0.9442

fig 3

Personal regression example

We extract 3000 datapoints from 13600 datapoints in this way:
1) take every 4th triplet of data, resulting in 3400 points
2) remove every 9th triplet (3400:9= 377 —> 3400-377 = 3023)
3) randomly remove 23 samples

The so obtained 3000 data-points are equally splitted into training validation and test sets.
Each set is defined by three variables: X1,X2 and Tnew.
Tnew represents an individual new non linear function, ie our target function, to be approximated by our neural network, and is built using the largest 5 digits of our student number in descending order: Tnew = (8T1+7T2+7T3+4T4+4T5)/30.
Figure.4 shows the plot of training set surface described by the so created X1 X2 and Tnew.

Figure.5 plot of transfer functions shows that
1) logsig and satlin have similar results and map most of the predictions to zero.
2) Purelin and tansig give better results i.e. low error.

Figure 6a shows the performance of different combinations of transfer functions and optimization algorithms on the training set:
1) LM performs equally regardless the transfer function used;
2) GD, BFG and CGP with satlin may overfit the training set (too much variance), same risk for GDA, BFG and CGF using logsig.
3) GDA and CGF with purelin, and LM may add too much bias during the training.

Figure 6b shows that
1) LM gives the lowest regardless the number of neurons used to built the network;
2) GDA gives the highest, especially when increasing the number of neurons used for building the network
3) none of the risk of misfitting (underfitting or overfitting) happens.

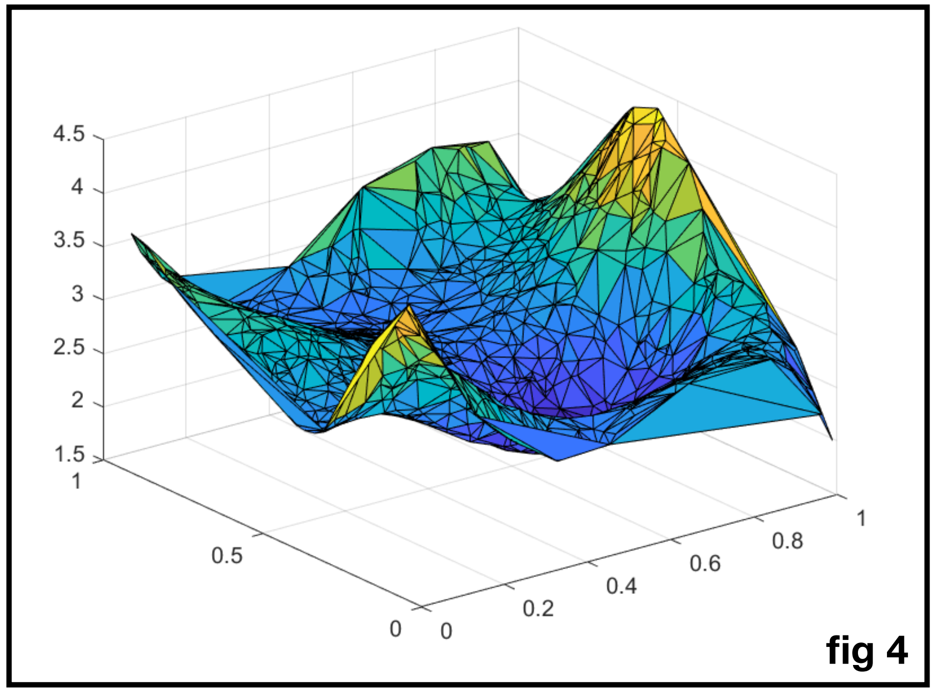


fig 4

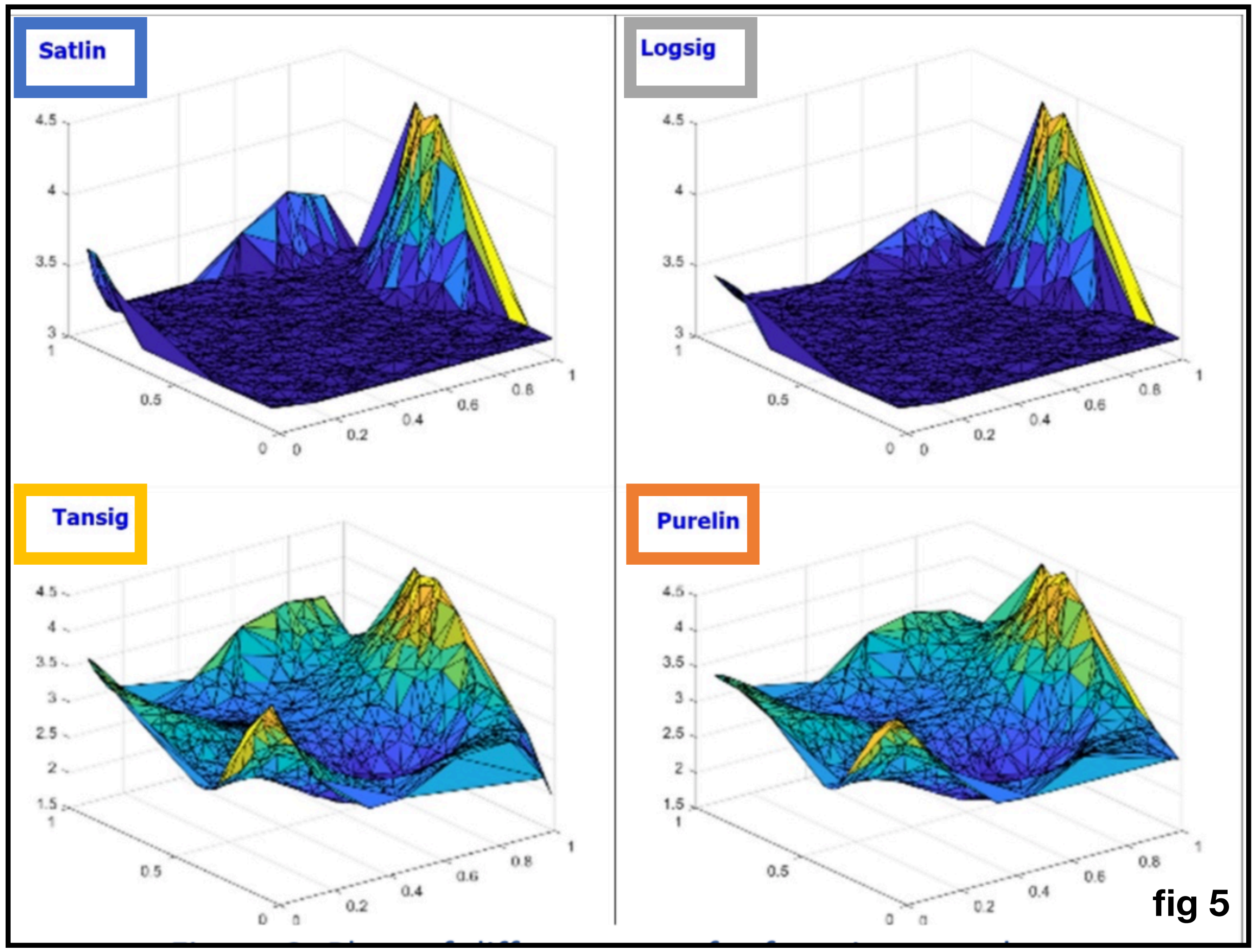
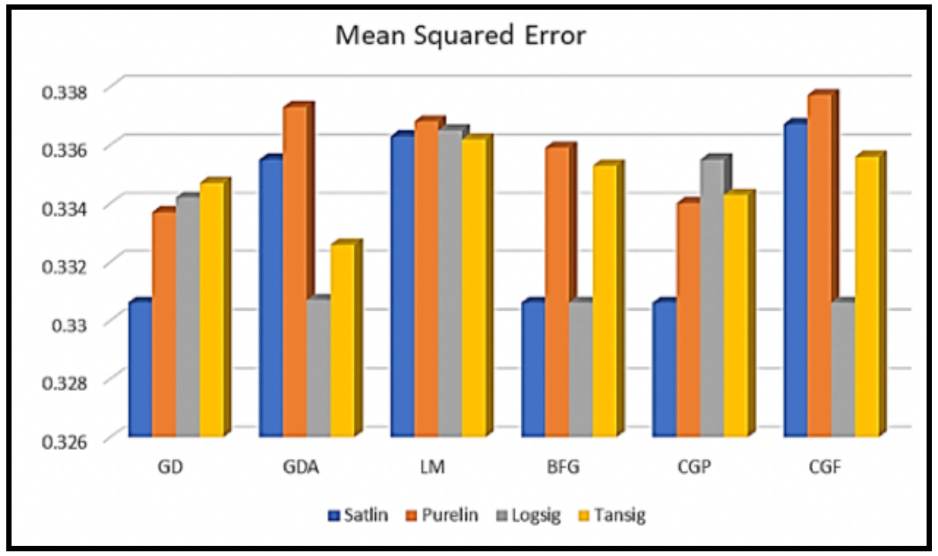


fig 5



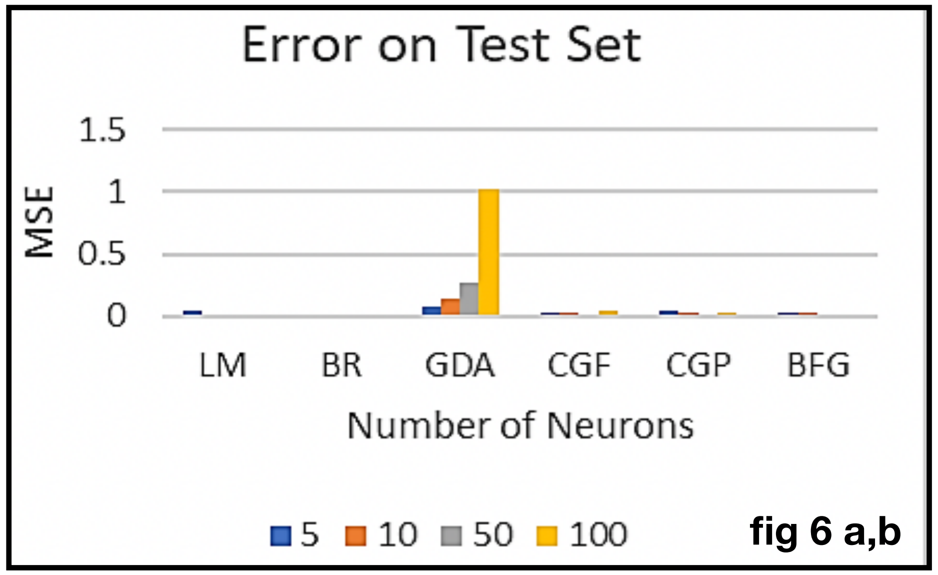


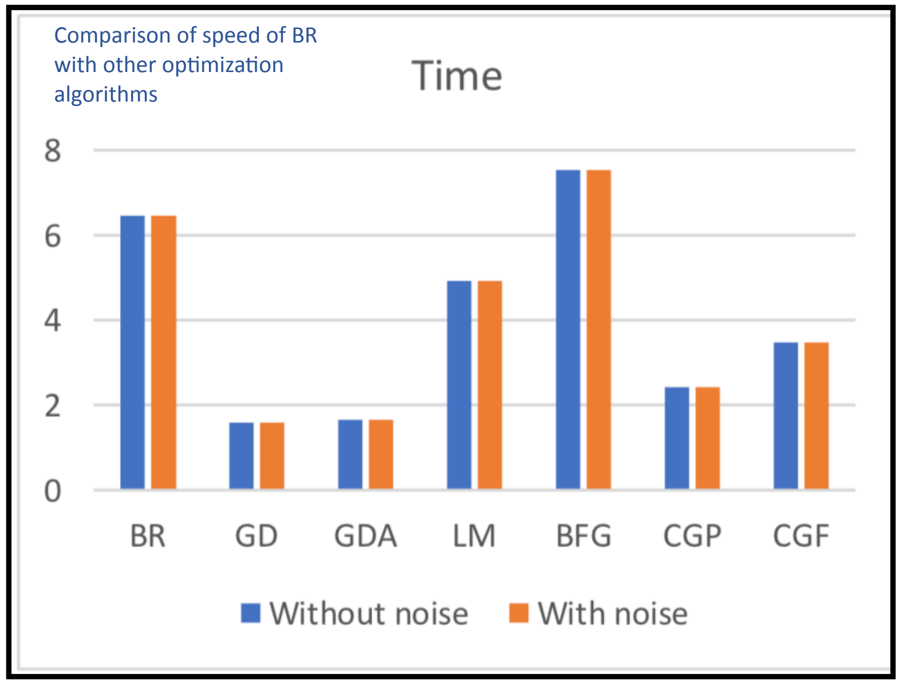
fig 6 a,b

Exercise 5: Bayesian Inference of Network: Hyperparameters

We use the Bayesian inference (BR) to optimize the performance of our network and compare its results with other optimization algorithms, for both noiseless and noisy data and use same network features as before.

Figure 7 shows the results of using BR for network training and using this network's weights and biases for all the other networks.
The graphs in Figure 7a show that:
1) BR takes much more time (almost 6.5 seconds) than other algorithms,
2) only BFG taking more time than BR.
3) All algorithms take same time t train on the noisy and noiseless data.

When it comes to performance (Figure 7b), we can observe that:
1) BR on the noiseless dataset performs well, with minimum MSE.
2) The error rate for BR on the noisy data is a little higher due to some overfitting.
3) For other algorithms, even though the weights and biases are changed now, the performance is almost the same as we have seen in the last section.
4) However, CGF, CGP, BFG, and LM have slightly higher MSE on the noisy data than the noiseless data suggesting overfitting.



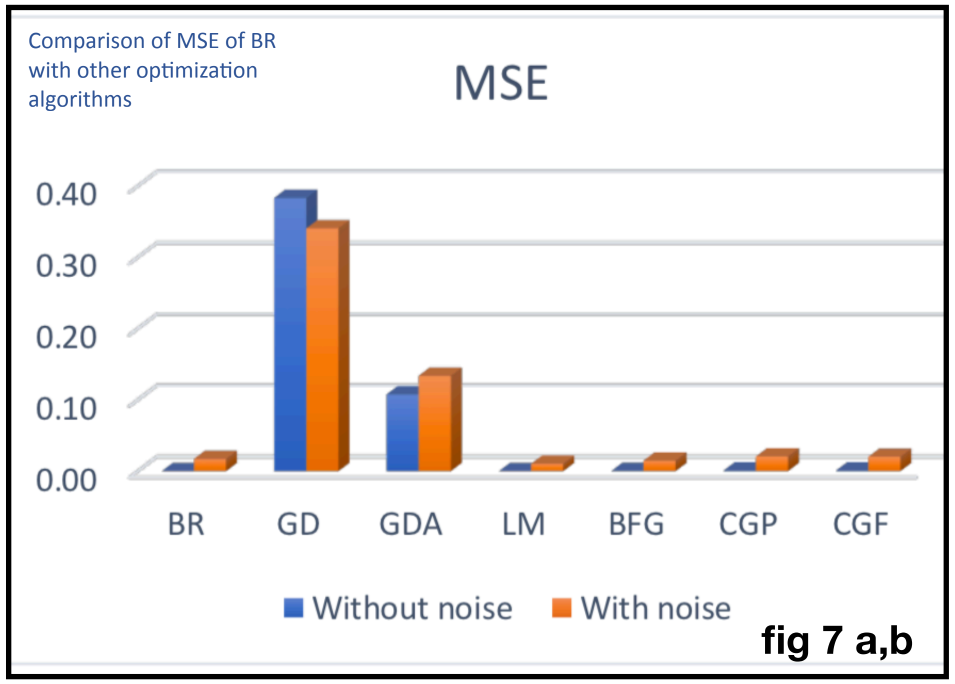
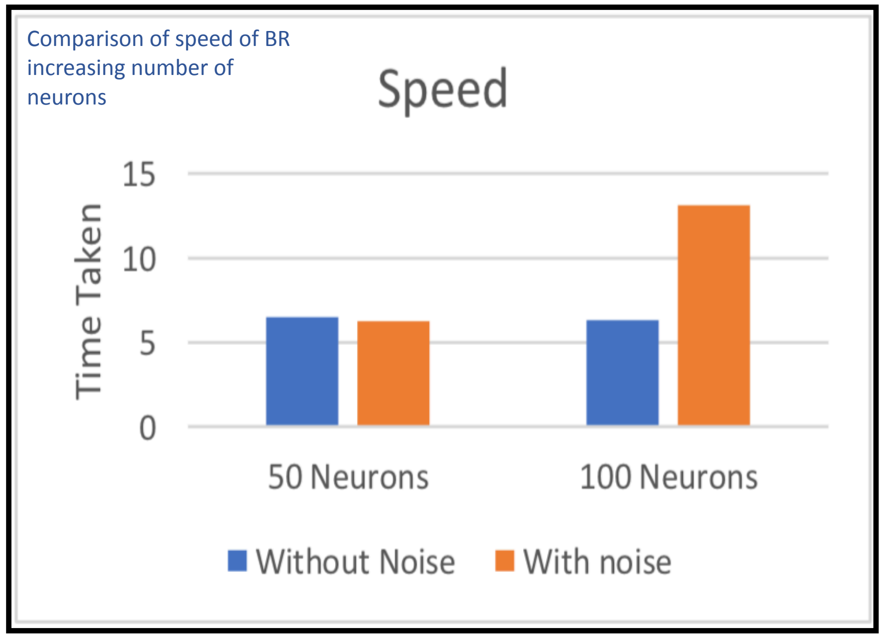


fig 7 a,b

Figure 8 shows the results of using BR for network training increasing the number of neurons: instead of 1 we now use 50 and 100 neurons.
Figure 8a shows that:
1) With 100 neurons the training time on the noisy data is almost double compared to that of noiseless data.

Figure 8b shows that:
2) MSE with 100 neurons on the noisy data is lesser than the MSE on the noisy data with 50 neurons.
3) The MSE and time for the noiseless data remains almost the same for both 50 and 100 neurons.



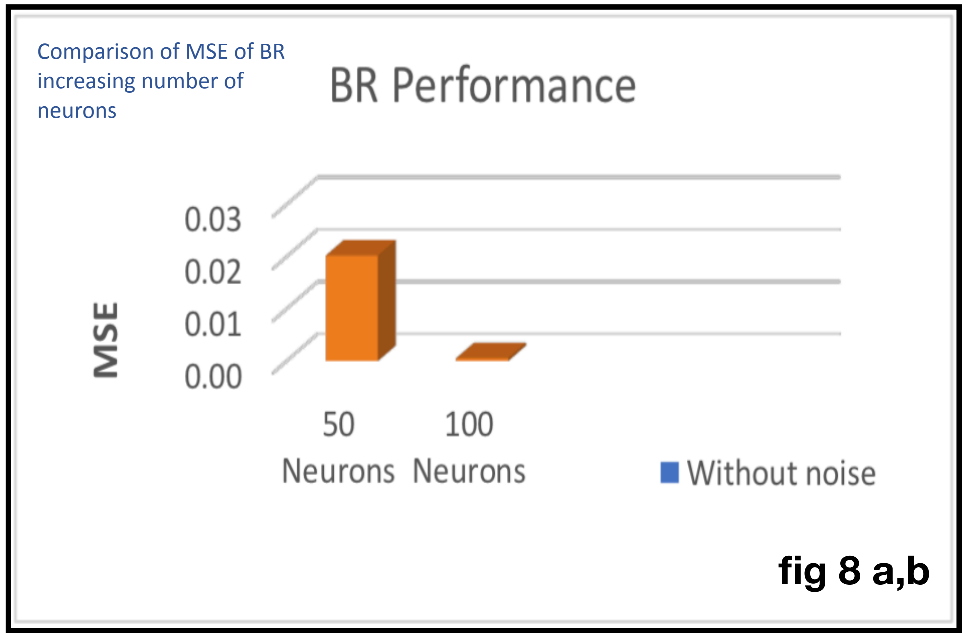


fig 8 a,b

So, in conclusion, increasing the number of neurons does improve the training efficiency of the network.