Homework 4 - Neural Network vs LogReg

Soheil Sadeghi Eshkevari

07-06-2018

1 Introduction

In this report, results of the last homework is going to be presented. In this project, we trained our machine to classify MNIST dataset, which is a well-known dataset of digit handwritings. For this purpose, both Logistic Regression and Neural Network were tried. For each, three different solvers, Stochastic Gradient Decent (SGD), SGD with Momentum and ADAM were exploited. Since all these optimization techniques are stochastic, various batch sizes were tried and results compared. In the next sections, results will be presented.

2 Logistic Regression Results

The first classifier that was tried was the Logistic Regression. This method is basically a binary classifier, thus to be able to train our dataset using it, we turned our labels to binary. For this purpose, label 5 is assigned to be +1 and other labels to be -1. Results are shown below:

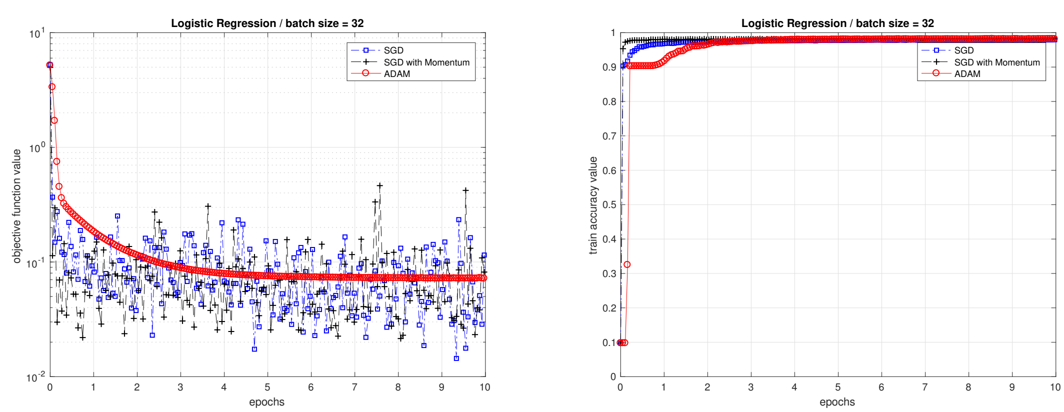

Figure 1: Performance plots vs. number of epochs with batch size 32

By comparing plots, following observations can be made. Less computational cost the problem needs when smaller batch sizes are set. In addition, with the setting I used for the training, ADAM is the most stable, while the least desirable one in terms of the fastest accuracy. Both SGD methods are very fast and suitable, however there show considerable fluctuations in their performance. This implies that the average of parameters can be a better alternative to get more stable results.

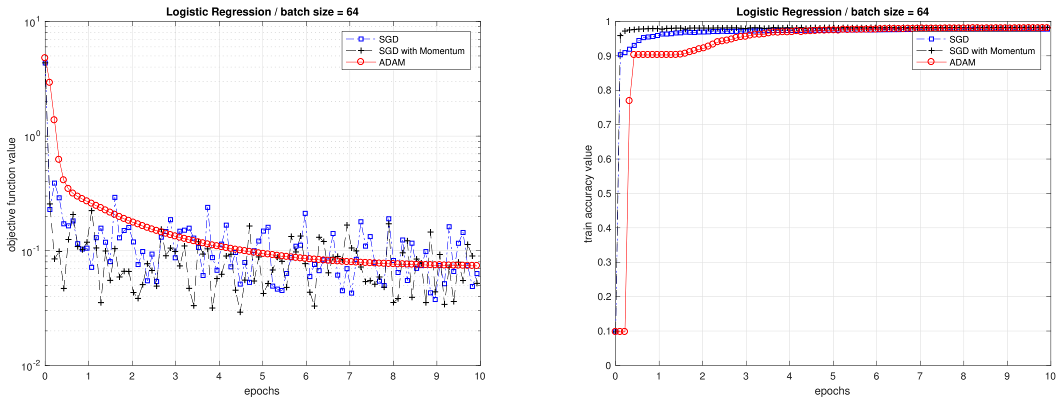

Figure 2: Performance plots vs. number of epochs with batch size 64

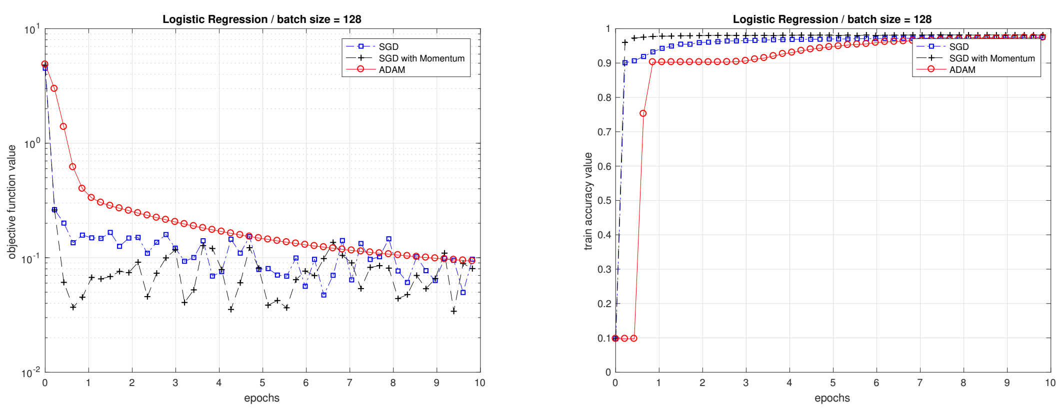

Figure 3: Performance plots vs. number of epochs with batch size 128

3 Neural Network Results, Multi-Label Classification

The next set of results belongs to the multi-class classification of the MNIST data using a 2 layer Neural Network. Sigmoid activation functions were used to connect layers. At the output layer, cross entropy function turns values to probabilities for each label. The number of nodes in the hidden layer also is set to be 40, based on both my own tuning and the suggested number from Mohammad and Mertcan's course project. As before, three different batch sizes were analyzed and results are shown below:

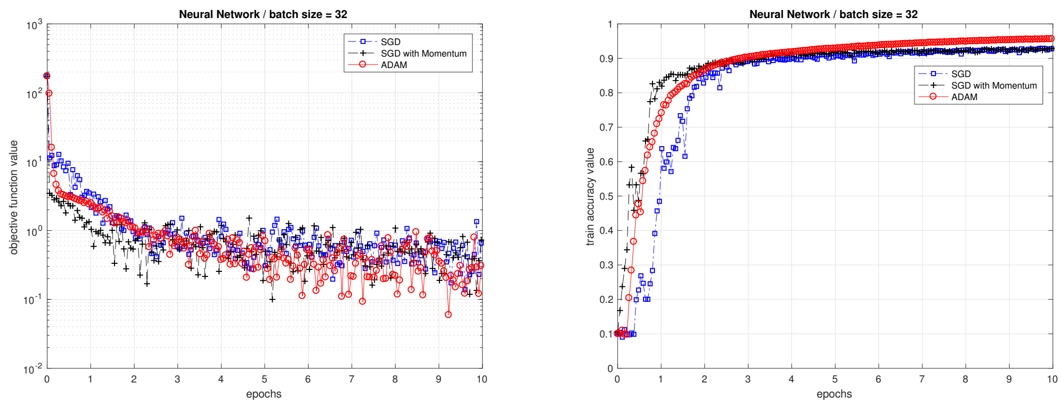

Figure 4: Performance plots vs. number of epochs with batch size 32

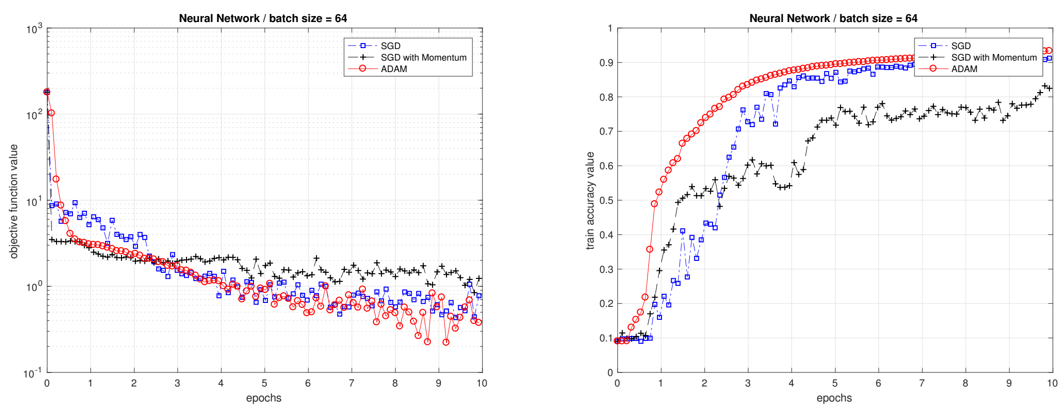

Figure 5: Performance plots vs. number of epochs with batch size 64

The most challenging part for NN training by SGD and SGD with momentum was to tune step sizes in a way that they converge to accurate results. In order to find the best values, first a fixed step was given and the best value for that was found (around 0.8). Next, a decaying step size in the format of $\frac{a}{b+k}$ was tuned to vary within 0.9 to 0.7. The step size was successful in the majority of cases, however, as it is noticeable in Figure 6, SGD with momentum in case of batch size equal to 128 was not able to get converged and resulted inaccurate predictions. In addition, by comparing different batch sizes, it can be interpreted that ADAM is obviously the dominant solver for NN since it is both stable and accurate. It also has no sensitivity to the step size and default settings worked appropriately.

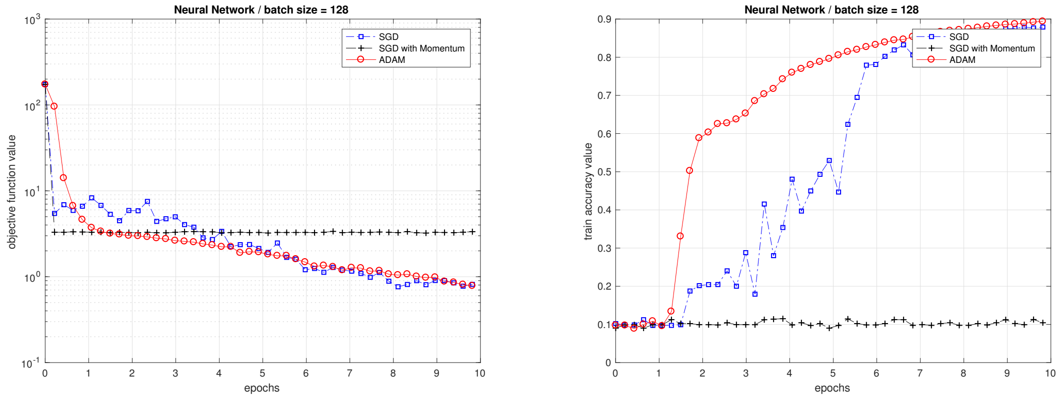

Figure 6: Performance plots vs. number of epochs with batch size 128

4 Fair comparison - Binary NN vs. Logistic Regression

As requested in the homework definition by Professor Scheinberg, in order to have a fair comparison between two methods, a binary classification problem with same logic explained for Logistic Regression was held using Neural Networks on MNIST dataset. The same analyses were done and results are presented hereafter:

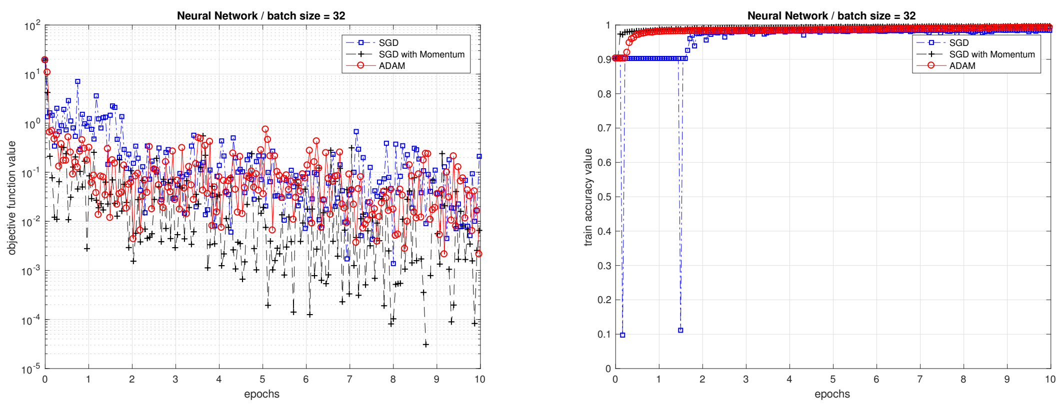

Figure 7: Performance plots vs. number of epochs with batch size 32

By comparing results, it can be deduced that NN is also very successful for binary classification of MNIST. The best accuracy of NN is slightly higher than the best accuracy derived from Logistic Regression. However, as before, for higher batch sizes, less desired behavior can be observed by NN. The most desired setting between these two methods is to use NN with batch size equal to 32 and using ADAM or SGD with momentum as the solver.

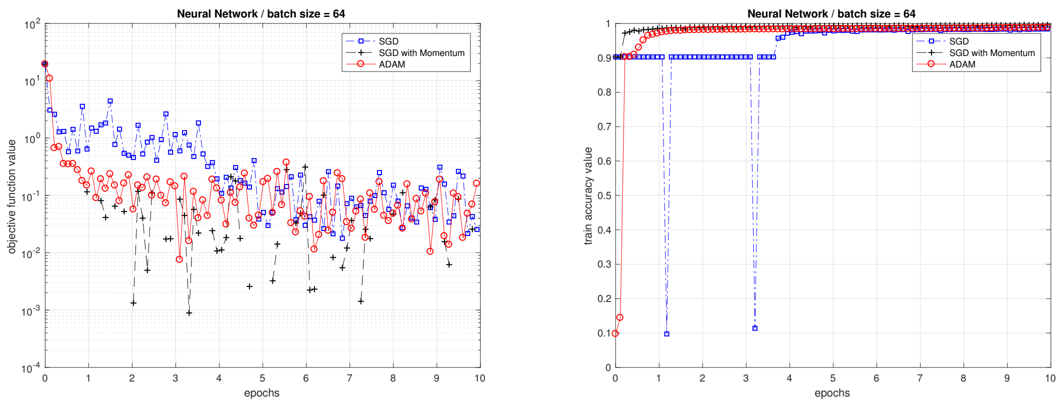

Figure 8: Performance plots vs. number of epochs with batch size 64

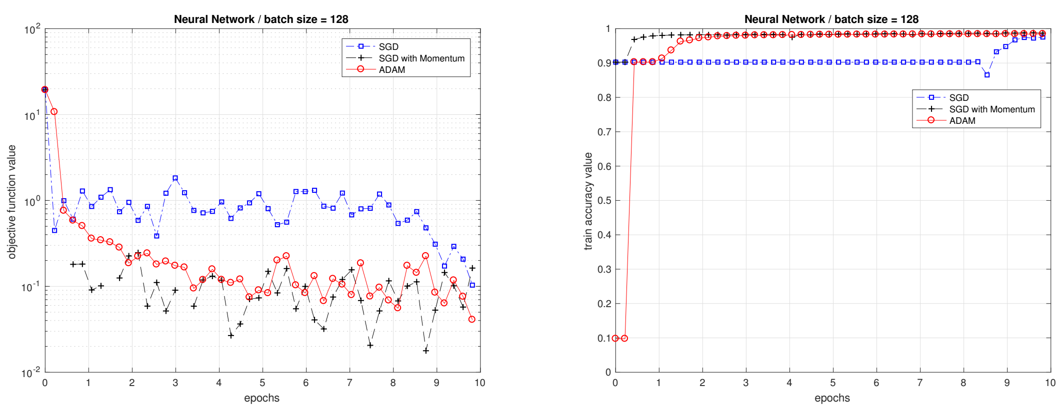

Figure 9: Performance plots vs. number of epochs with batch size 128

5 Code

In this section, the main body of my NN code for this homework is presented. However, subroutines and associated files will be sent in a package. Please note that a similar file exists for Logistic Regression and can be found in the package.

```
% DATABASE 4: mnist.mat (always run this section in advance, for all for four algorithms)

clc;
clear;

rng shuffle

data_1 = load('new_data/mnist_mult.mat');

X_temp = data_1.TrainX;
Y = data_1.TrainY;
Xt_temp = data_1.TestX;
Y_t = data_1.TestY;

% fair comparison (Binary class NN) --------
Y_temp = zeros(size(Y,1),2);
Y_temp(:,1) = Y(:,5)==1;
Y_temp(:,2) = Y(:,5)==0;
Y = Y_temp;

Y_temp = zeros(size(Y_t,1),2);
Y_temp(:,1) = Y_t(:,5)==1;
Y_temp(:,2) = Y_t(:,5)==0;
Y_t = Y_temp;
Y_temp = [];
% ------------------------------------------

X = zeros(60000,28*28);
for i = 1:60000
    temp = squeeze(X_temp(:,:,i));
    X(i,:) = temp(:);
end

X_t = zeros(10000,28*28);
for i = 1:10000
    temp = squeeze(Xt_temp(:,:,i));
    X_t(i,:) = temp(:);
end

%% Scaling and parameters

for i = 1:size(X,1)
    X(i,:) = X(i,:)/norm(X(i,:),2);
end
for i = 1:size(X_t,1)
    X_t(i,:) = X_t(i,:)/norm(X_t(i,:),2);
end

input_layer_size  = 28*28;% 28x28 Input Images of Digits
hidden_layer_size = 40;   % 40 hidden units
num_labels = 2;           % 10 labels, from 1 to 10
                          % (note that we have mapped "0" to label 10)
lambda = 0.0;

% w_1 = rand(hidden_layer_size, (input_layer_size + 1));
% w_2 = rand(num_labels, (hidden_layer_size + 1));
```

```
w_1 = rand(hidden_layer_size, (input_layer_size));
w_2 = rand(num_labels, (hidden_layer_size));
nn_params = [w_1(:); w_2(:)];
% a_2 = 1./(1+X'*w_1);
% p = 1./(1+a_2*w_2);
% hs = -log(exp(p)./sum(exp(p),2));

batch_size = 128; % default = 50
epochs = 10;
iter = ceil(epochs*size(X,1)/batch_size);

%% solver (ADAM)

alfa = 0.002; % 0.002
beta_1 = 0.9;
beta_2 = 0.999;
eps = 1e-8;
m = zeros(numel(nn_params),1);
v = zeros(numel(nn_params),1);
t = 0;
thresh = 1e-2;

grad = 1;
J_tr_ADAM = zeros(iter,1);
J_ts_ADAM = zeros(iter,1);
nn_params_ADAM = zeros(iter+1,numel(nn_params));
nn_params_ADAM(1,:) = nn_params;

for j = 1:iter
    t = t + 1;
    Sk = randi(size(X,1),1,batch_size);
    [J_tr_ADAM(j), grad] = nnCostFunction(nn_params_ADAM(j,:), ...
                                input_layer_size, ...
                                hidden_layer_size, ...
                                num_labels, ...
                                X(Sk,:), Y(Sk,:), lambda);
    m = beta_1*m + (1-beta_1)*grad;
    v = beta_2*v + (1-beta_2)*(grad.^2);
    m_hat = m/(1-beta_1^t);
    v_hat = v/(1-beta_2^t);
    nn_params_ADAM(j+1,:) = nn_params_ADAM(j,:) - (alfa * m_hat./(sqrt(v_hat)+eps))';
    J_ts_ADAM(j) = nnCost(nn_params_ADAM(j+1,:), ...
                                input_layer_size, ...
                                hidden_layer_size, ...
                                num_labels, ...
                                X_t, ...
                                Y_t, lambda);
                            j
%     obj_fnc = J_tr_ADAM(j)
%     norm(grad,2)
end

% plot(1:20:iter,[J_tr_ADAM(1:20:end) J_ts_ADAM(1:20:end)]);

%% solver (SGD)

a = 28350;
b = 30500;
teta = 0.95;

k = 1:iter;
nu = a./(b+k);

J_tr_SGD = zeros(iter,1);
J_ts_SGD = zeros(iter,1);
```

```
nn_params_SGD = zeros(iter+1,numel(nn_params));
nn_params_SGD(1,:) = nn_params;

v = zeros(length(nn_params),1);
v_prev = zeros(length(nn_params),1);

for j = 1:iter
    Sk = randi(size(X,1),1,batch_size);
    [J_tr_SGD(j), grad] = nnCostFunction(nn_params_SGD(j,:), ...
                                    input_layer_size, ...
                                    hidden_layer_size, ...
                                    num_labels, ...
                                    X(Sk,:), Y(Sk,:), lambda);
    v = teta * v_prev + grad;
    v_prev = v;
    nn_params_SGD(j+1,:) = nn_params_SGD(j,:) - nu(j) * grad';
%     nn_params = nn_params - nu(j) * grad;
%     nn_params = nn_params - 0.1 * v;
    J_ts_SGD(j) = nnCost(nn_params_SGD(j+1,:), ...
                                    input_layer_size, ...
                                    hidden_layer_size, ...
                                    num_labels, ...
                                    X_t, ...
                                    Y_t, lambda);
                                j
%     J_tr_SGD(j)
%     norm(grad,2)
end

% plot(J_ts_SGD(2:j)); hold on; plot(J_tr_SGD(2:j));

%% solver (SGD + Momentum)

a = 28350;
b = 30500;
teta = 0.95;

k = 1:iter;
nu = a./(b+k);

J_tr_SGDM = zeros(iter,1);
J_ts_SGDM = zeros(iter,1);
nn_params_SGDM = zeros(iter+1,numel(nn_params));
nn_params_SGDM(1,:) = nn_params;
v = zeros(length(nn_params),1);
v_prev = zeros(length(nn_params),1);

for j = 1:iter
    Sk = randi(size(X,1),1,batch_size);
    [J_tr_SGDM(j), grad] = nnCostFunction(nn_params_SGDM(j,:), ...
                                    input_layer_size, ...
                                    hidden_layer_size, ...
                                    num_labels, ...
                                    X(Sk,:), Y(Sk,:), lambda);
    v = teta * v_prev + grad;
    v_prev = v;
%     nn_params = nn_params - nu(j) * grad;
%     nn_params = nn_params - nu(j) * grad;
    nn_params_SGDM(j+1,:) = nn_params_SGDM(j,:) - nu(j) * v';
    J_ts_SGDM(j) = nnCost(nn_params_SGDM(j+1,:), ...
                                    input_layer_size, ...
                                    hidden_layer_size, ...
                                    num_labels, ...
                                    X_t, ...
                                    Y_t, lambda);
```

```
                                j
%     J_tr_SGDM(j)
%     norm(grad,2)
end

% plot(J_ts_SGDM(2:j)); hold on; plot(J_tr_SGDM(2:j));

%% plots
step = 100;
% function in three methods
semilogy((1:step:iter)*batch_size/size(X,1),J_tr_SGD(1:step:end), '-.bs'); hold on
semilogy((1:step:iter)*batch_size/size(X,1),J_tr_SGDM(1:step:end), '--k+'); hold on
semilogy((1:step:iter)*batch_size/size(X,1),J_tr_ADAM(1:step:end), '-ro');
grid on
legend('SGD','SGD with Momentum','ADAM');
xlabel('epochs');
ylabel('objective function value');
title(['Neural Network / batch size = ',num2str(batch_size)])

% accuracy in three methods
y_vec = zeros(size(Y,1),1);
for i = 1:size(Y,1)
    y_vec(i) = find(Y(i,:)==1);
end

pred_SGD = zeros(1,length(1:step:iter));
pred_SGDM = zeros(1,length(1:step:iter));
pred_ADAM = zeros(1,length(1:step:iter));
for i = 1:step:iter
    pred_SGD(i) = mean(double(predict(nn_params_SGD(i+1,:), ...
        X,hidden_layer_size,input_layer_size,num_labels) == y_vec));
    pred_SGDM(i) = mean(double(predict(nn_params_SGDM(i+1,:), ...
        X,hidden_layer_size,input_layer_size,num_labels) == y_vec));
    pred_ADAM(i) = mean(double(predict(nn_params_ADAM(i+1,:), ...
        X,hidden_layer_size,input_layer_size,num_labels) == y_vec));
end
figure;
plot((1:step:iter)*batch_size/size(X,1),pred_SGD(1:step:end), '-.bs'); hold on
plot((1:step:iter)*batch_size/size(X,1),pred_SGDM(1:step:end), '--k+'); hold on
plot((1:step:iter)*batch_size/size(X,1),pred_ADAM(1:step:end), '-ro');
grid on
legend('SGD','SGD with Momentum','ADAM');
xlabel('epochs');
ylabel('train accuracy value');
title(['Neural Network / batch size = ',num2str(batch_size)])
```

6 Conclusions

In this homework, the main catch was to have a hands-on experience on coding some *off the shelf* methods and solvers and try to play with parameters to get a deeper understanding about these, rather than just superficially use them as blackbox toolsuites.