WEEK 2 ASSIGNMENTS
1)Linear Regression Assignment
2)Function Approximation Assignment
3)MNIST MLP Assignment

Linear Regression Assignment

1. Adapt the Examples_LinearRegression.py script to determine a suitable number of training cycles required for the fit to converge to a stable value of the cost.[1] Note this down, and provide a plot of the epoch vs cost to justify your solution.
2. Adapt the script to investigate the fit convergence behaviour for the following scenarios and summarize the results (including any plots that you deem to be relevant) in a short report (max 2 pages):

c	0	0	0
m	1	10	100

1.1)

```
-------------------------------------------
Number of examples to generate =  100
Learning rate            alpha =  0.005
Number of training epochs      =  100
-------------------------------------------
0 [ 0.90120286] [ 1.18517733] 13.0573 1
1 [ 0.92277265] [ 1.03873801] 10.5354 2.52191162109375
2 [ 1.00496361] [ 1.02853399] 9.2435 1.29185390472412

70 [ 1.98487055] [ 0.52668631] 0.993095 9.769201278686523e-05
Optimisation process has finished.  The optimal values of parameters are:
  m =  [ 1.98487055]  [input value =  2.0 ]
  c =  [ 0.52668631]  [input value =  0.5 ]
  loss function for the optimal parameters =  0.993095
```

max number of epochs

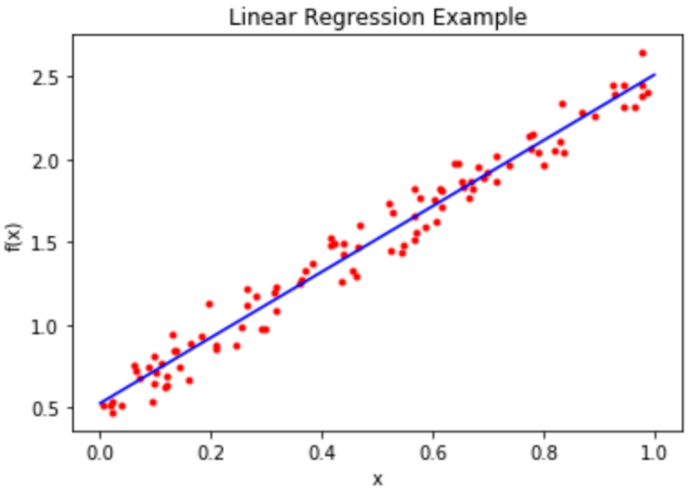
sufficient number of epochs

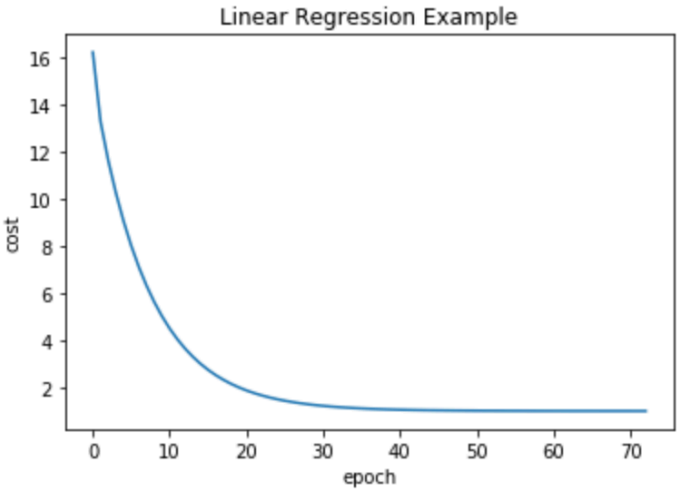
difference of loss from one epoch to the following one

m (gradient)

c (intercept)

loss function
(chi squared= (y - y_) * (y - y_))



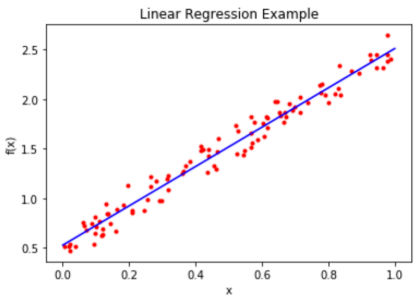


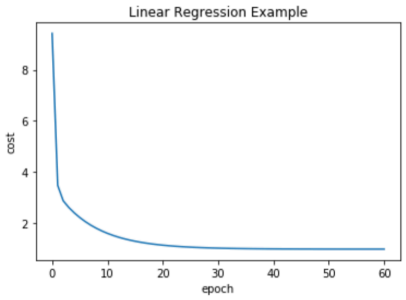
I determined 70 as a suitable number of fit iterations required for the fit to converge to the minimum X2 value because, set a tolerance parameter to 0.0001 as the difference from one epoch to the next one under which it's possible to stop training the optimizer, at the 70th cycle the difference of the cost function was 0.00001 < 0.0001.

1.2)

with c= 0, m= 1

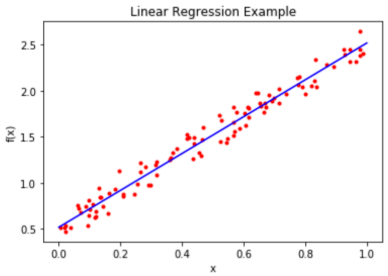
```
60 1.98528 0.526477 0.993035 8.857250213623047e-05
Optimisation process has finished.  The optimal values of parameters are:
  m =  1.98528  [input value =  2.0 ]
  c =  0.526477  [input value =  0.5 ]
  loss function for the optimal parameters =  0.993035
```

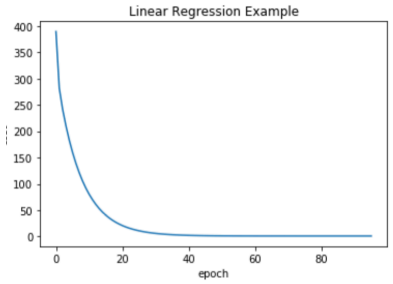




with c= 0, m= 10

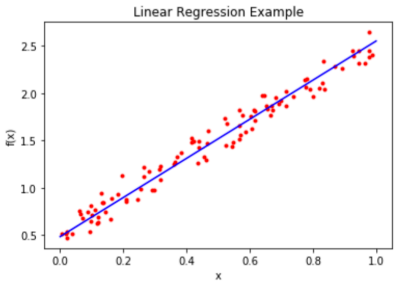
```
95 2.0021 0.517955 0.993034 8.863210678100586e-05
Optimisation process has finished.  The optimal values of parameters are:
  m =  2.0021  [input value =  2.0 ]
  c =  0.517955  [input value =  0.5 ]
  loss function for the optimal parameters =  0.993034
```

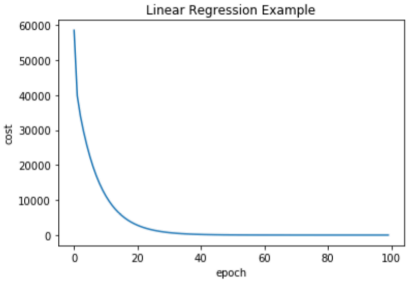




with c= 0, m= 100

```
99 2.06936 0.483871 1.04068 0.007184386253356934
Optimisation process has finished.  The optimal values of parameters are:
  m =  2.06936  [input value =  2.0 ]
  c =  0.483871  [input value =  0.5 ]
  loss function for the optimal parameters =  1.04068
```





What I noticed is that when given c=0 and m=1, ie when initial parameter values where more similar to the actual one (m= gradient= 2.0; c= intercept = 0.5), the fit to convergence occurred in a less number of epochs.

Function Approximation Assignment

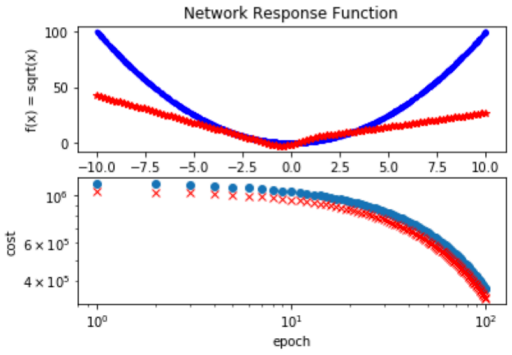
1. Write an MLP with 1 input node, 1 output nodes and 2 hidden layers, each with 50 hidden nodes. Compare the training performance for the 2 layer MLP to that of the original 1 layer MLP. Hint – start from `Example_FunctionApproximator.py`.
2. Make a plot to compare the cost evolution as a function of training epoch for (1) with / without the use of dropout for a keep probability of 50%.
3. Make a plot to compare the cost evolution as a function of training epoch for batch training vs using all of the training events in one go.
4. Make a plot to show the combined performance of using both batch training and dropout to explore the potential for using both methods.
5. Summarise these results in a short report (2 pages maximum) for submission along with the Python code used to make each of the plots.

2.1 and 2.2)

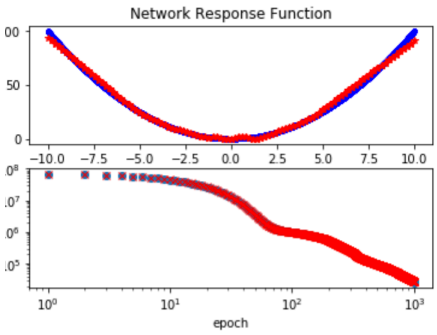
```
learning_rate    = 0.001
training_epochs  = 100
min_x = -10
max_x = 10
Ngen   = 1000

n_input     = 1
n_classes   = 1
n_hidden_1 = 50
```

Starting from an MLP with 1000 random generated input values, one hidden layer with 50 hidden nodes and one output and set hyper parameters to be 0.001 for the learning rate, 100 for the number of epochs, I wrote an MLP with the same hyper parameters but with two hidden layers, each consisting of 50 hidden nodes.

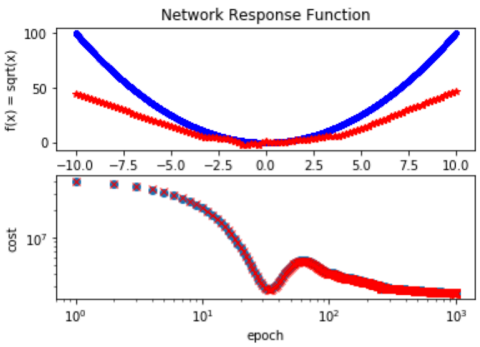
I then compared the 1 hidden layer model given with the one implemented with a second layer and a third model which combined the second model architecture with a drop out regularization, with a drop out keep probability of a 0.5 (ie: compromise the model randomly in different epochs of the training by removing half of the units from the network to limit overfitting. When evaluating predictions with the validation or unseen data the full network.



1H, no dropout



2H, no dropout



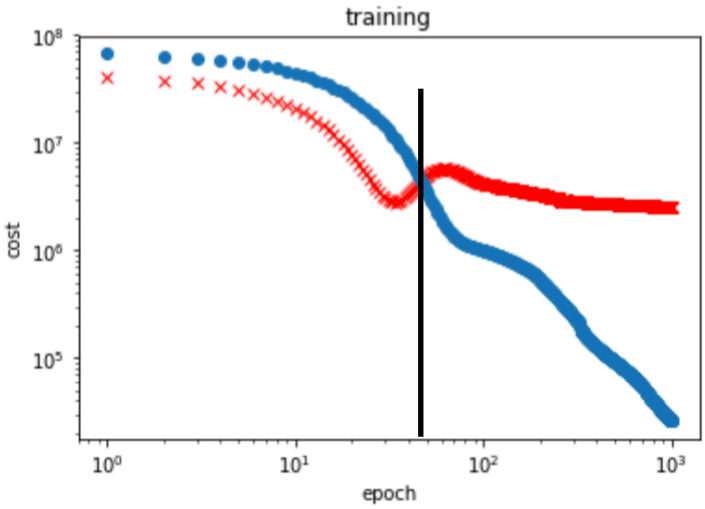
2H, 0.5 dropout

While adding a layer critically improved the model performances from the very first epoch, adding the dropout regularization initially weakened the model, but when increasing the number of epoch it actually surpassed it.

That occurs because even though dropout is a pragmatic way to mitigate overfitting (the whole model will be trained on a sub-sample of the data in the hope that the effect of statistical fluctuations will be limited), dropout can be detrimental for:

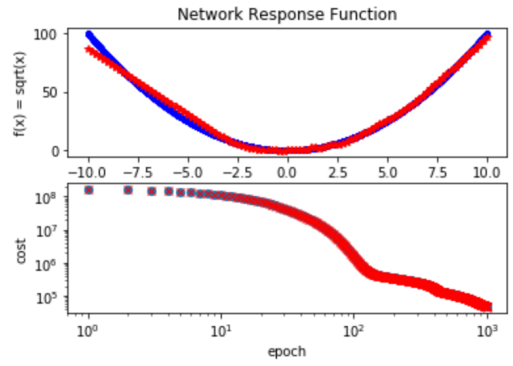
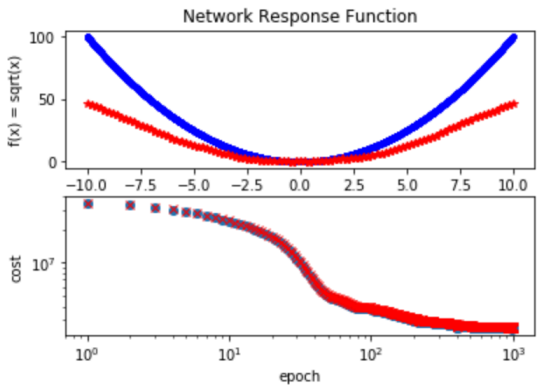
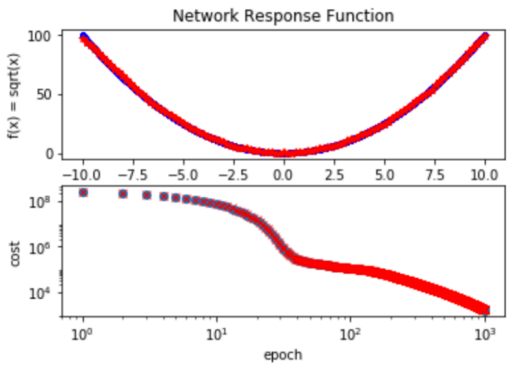
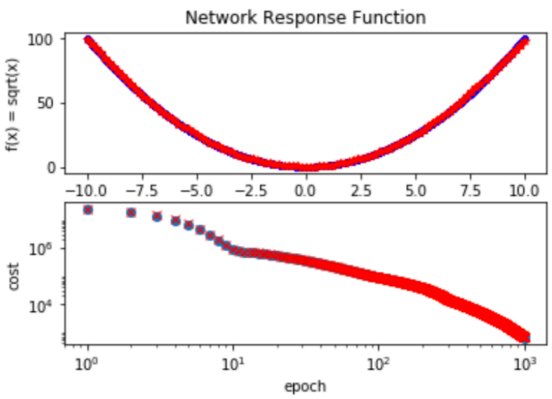
a) small training samples
b) data trained with a few epochs, because for each epoch the model would learn from smaller samples

After all, 1000 training samples should be enough to benefit from the drop out technique, yet its contribution can be better appreciated only after increasing the number of epochs.



Even though in the first epochs the model with 2 hiddens and no drop out (the blue in the above figure)performed better, from epoch 40 ca the model with the batch learning (the red in the above figure) performed better.

2.3, 2.4 and 2.5)

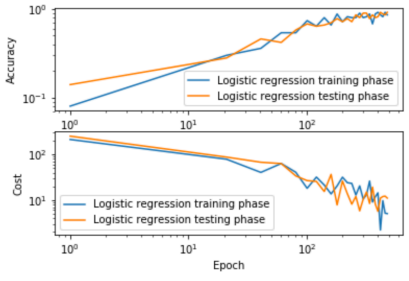
	no dropout	0.5 dropout
no batches		
11 batches		

Using a batch learning together with the 0.5 dropout seems to make things work better, because the mini batches compensate the few number of epochs.
Furthermore using batch learning (to better estimate the gradient for minimization) leads to a faster convergence. This happens because for many problems there are clusters of similar training examples.

MNIST MLP Assignment

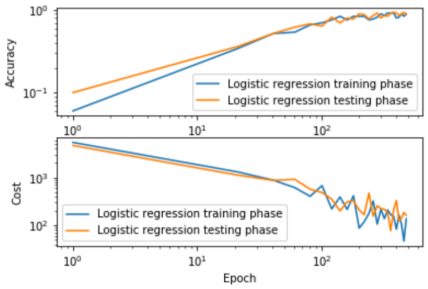
1. Starting from `Example_MNIST_MLP.py`, adapt the script to allow for at least 3 layers in the network. Compare the accuracy obtained when training the network for 500 training epochs for the 1 layer configuration provided and your new configuration.
2. Systematically explore how the learning rate affects performance; documenting in particular the cost and accuracy as a function of training epoch.
3. Systematically explore how the drop out keep probability affects performance; documenting in particular the cost and accuracy as a function of training epoch.
4. Reflect on your results in the context of the function approximation example and not any specific observations you make and conclusions that you are able to draw regarding training.
5. Summarise these results in a short report (2 pages maximum) for submission along with the Python code used to make each of the plots.

3.1)



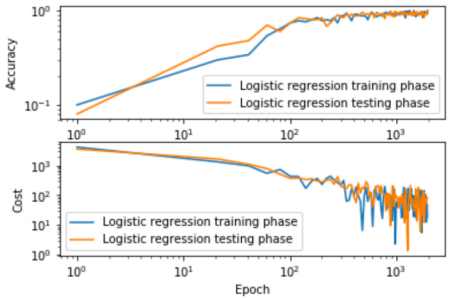

```
step 480, training accuracy 0.86    test accuracy 0.92
Training phase finished
 accuracy for the train set examples       =  0.86
 accuracy for the test set examples        =  0.92
```

500 epochs 1 hidden




```
step 480, training accuracy 0.9    test accuracy 0.88
Training phase finished
 accuracy for the train set examples       =  0.9
 accuracy for the test set examples        =  0.88
```

500 epochs 3 hiddens

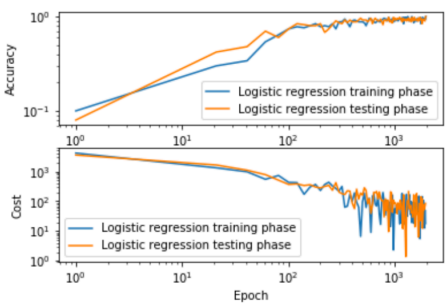



```
step 1980, training accuracy 1    test accuracy 1
Training phase finished
 accuracy for the train set examples       =  1.0
 accuracy for the test set examples        =  1.0
```

2000 epochs 3 hiddens

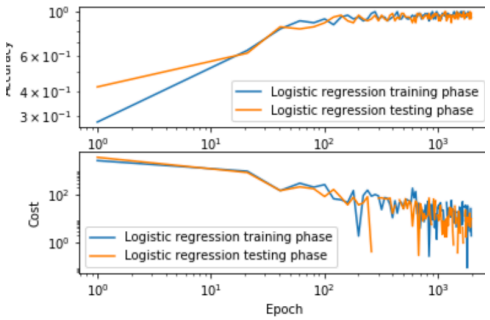
Comparing the accuracy obtained when training the network for 500 training epochs for the 1 layer configuration provided and the accuracy obtained when training the two networks with 3 layers for 500 and 2000 epochs I found that increasing the number of layers and epochs improved the results on the test set, but at a certain number of epochs oscillation in the training and testing cost becomes more severe.

3.2)



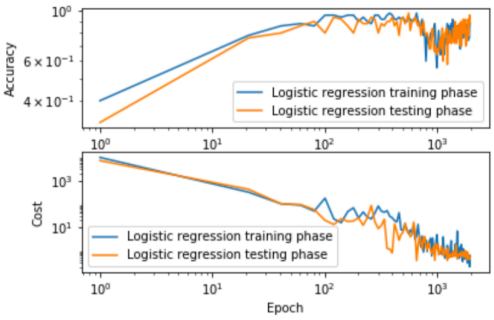

```
step 1980, training accuracy 1    test accuracy 1
Training phase finished
 accuracy for the train set examples       =  1.0
 accuracy for the test set examples        =  1.0
```

2000 epochs 3 hiddens
lr = 0.001




```
step 1980, training accuracy 0.94    test accuracy 0.98
Training phase finished
 accuracy for the train set examples       =  0.94
 accuracy for the test set examples        =  0.98
```

2000 epochs 3 hiddens
lr = 0.01



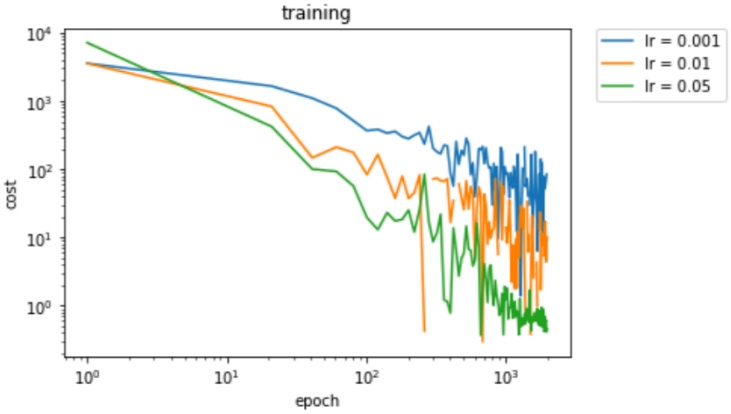

```
step 1980, training accuracy 0.88    test accuracy 0.88
Training phase finished
 accuracy for the train set examples       =  0.88
 accuracy for the test set examples        =  0.88
```

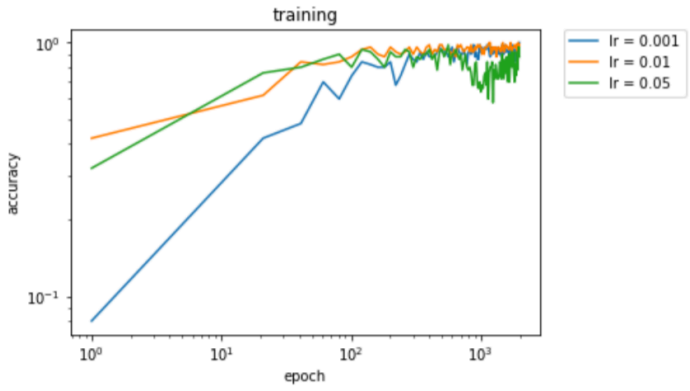
2000 epochs 3 hiddens
lr = 0.5

The learning rate is a parameter that tunes the step size taken in the optimisation process.

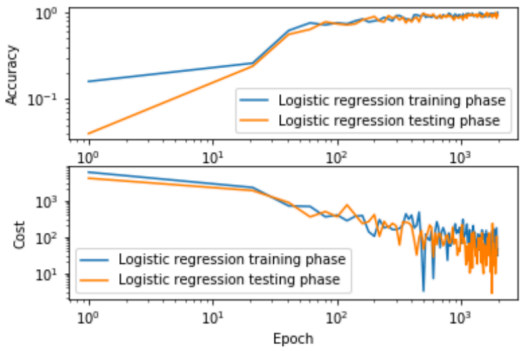
Small values of the learning rate will allow the optimisation algorithm to converge on a good approximation to the optimal solution - but this may take a long time.
Large values of this parameter can allow for faster convergence to the region where the optimal solution lies; but there is a limit to how close one can converge to that solution.

The second model is slightly overfitting the data, compared to the first one (accuracy on train is better, but on test is worse); the third just jumps too big to reach a good minima.
So I think that the optimal learning rate will be 0.001.



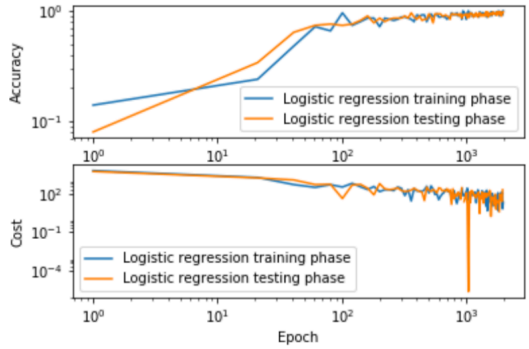


3.3, 3.4 and 3.5)



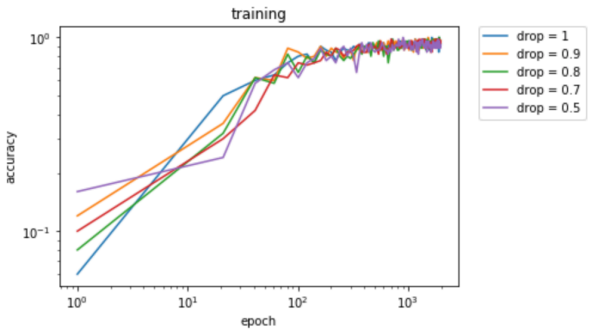
3H, epochs: 2000, lr: 0.001, drop: 1

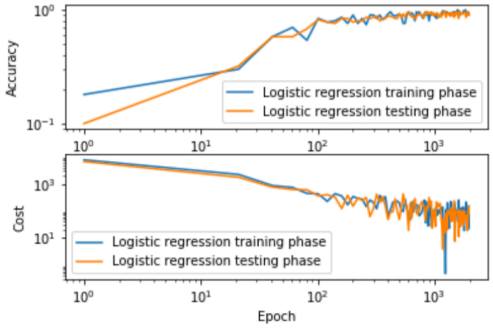
```
step 1980, training accuracy 1    test accuracy 0.94
Training phase finished
 accuracy for the train set examples       =  1.0
 accuracy for the test set examples        =  0.94
```



3H, epochs: 2000, lr: 0.001, drop: 0.9

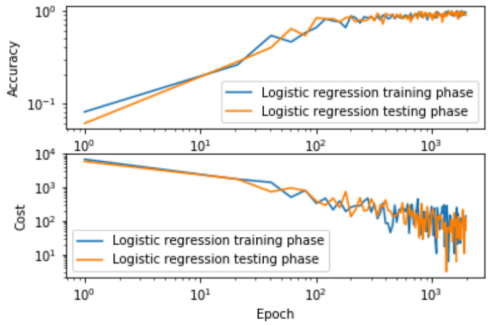
```
step 1980, training accuracy 1    test accuracy 0.98
Training phase finished
 accuracy for the train set examples       =  1.0
 accuracy for the test set examples        =  0.98
```





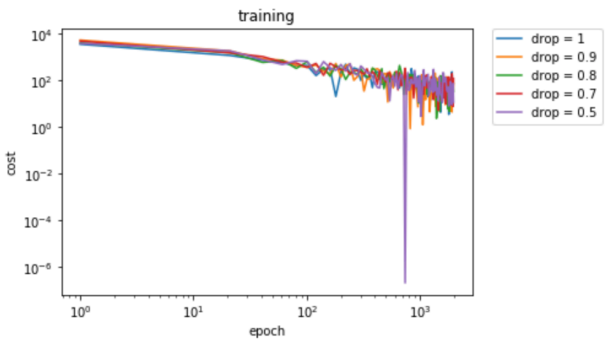
3H, epochs: 2000, lr: 0.001, drop: 0.7

```
step 1980, training accuracy 0.92    test accuracy 0.9
Training phase finished
 accuracy for the train set examples       =  0.92
 accuracy for the test set examples        =  0.9
```



3H, epochs: 2000, lr: 0.001, drop: 0.5

```
step 1980, training accuracy 0.96    test accuracy 0.9
Training phase finished
 accuracy for the train set examples       =  0.96
 accuracy for the test set examples        =  0.9
```



No significant differences in performance were shown in modifying the percentage of dropouts. This could be probably because we trained our model on the best model we had: learning rate 0.001, 3 hiddens and 2000 epochs!