Arnau Colom - U137471
David Moreno - U114148

Programming Exercise 2

Arnau Colom & David Moreno

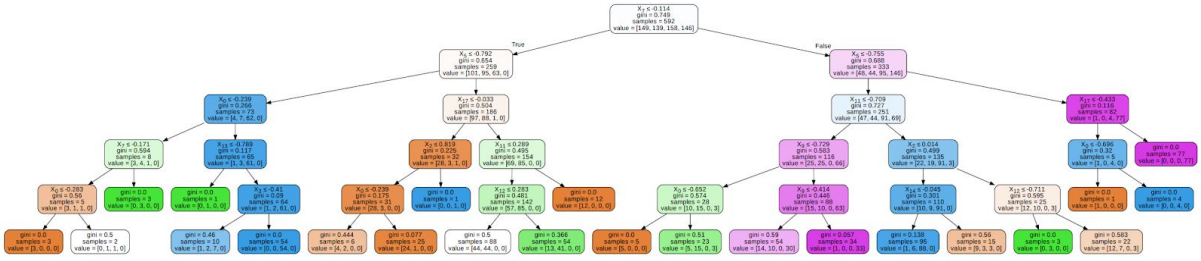
Decision Trees

Partition the dataset into a training and a testing set. Run a decision tree learning algorithm using the training set. Test the decision tree on the testing dataset and report the total classification error (i.e. 0/1 error). Repeat the experiment with a different partition. Plot the resulting trees. Are they very similar, or very different? Explain why.
Advice: it can be convenient to set a maximum depth for the tree.

```
Data used: vehicle
Classes: 4
Features: 18
Data-size: 846
```

We have chosen the vehicle dataset to test one of the decision trees' nice properties, it's good performance with not so big datasets.

We started from what we think it's a convenient data splitting, 70% of the data used for the training, and 30% for the testing. Also we set up the max_depth to 5, to be able to understand the tree organization better.



Decision Tree with 0.3 testing data size

Even though in the figure above it's not possible to see the data splitting, neither the gini nor the other parameters, we can observe that it is a quite balanced tree, and also, by the values, it is compensated in the first branches regarding the data size.

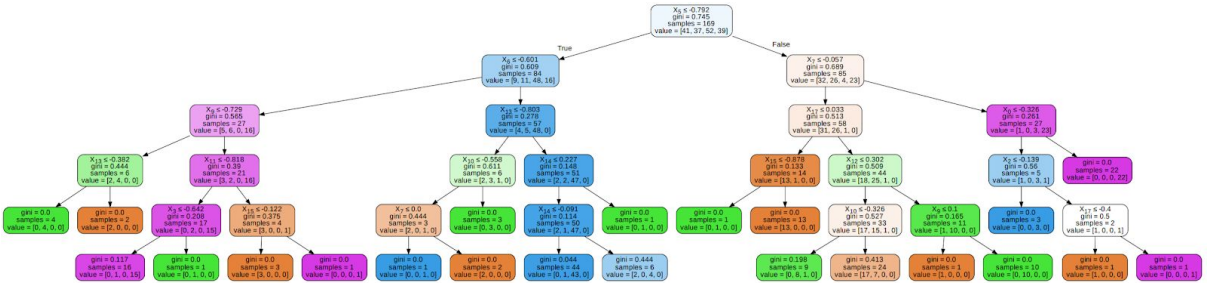
Scoring this tree with the resting data, the testing data, we obtain:

```
Accuracy:  0.65748031496063
Error:  0.34251968503937
```

Which is not as good as we expected.

Arnau Colom - U137471
David Moreno - U114148

In a second try, we have trained the model with a 20% of the data, and the resulting tree is the following

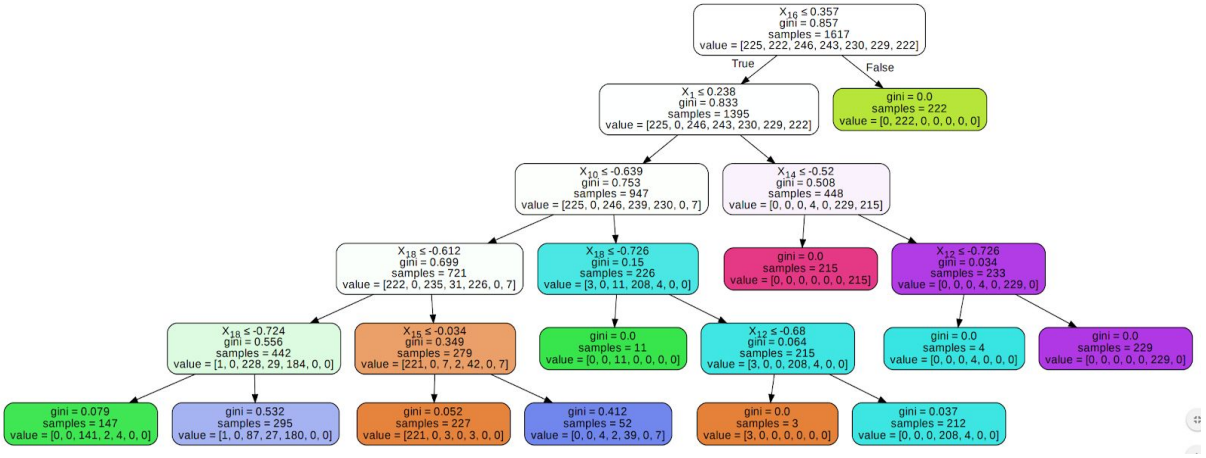


Decision Tree with 0.8 testing data size

```
Accuracy:  0.6366322008862629
Error:  0.36336779911373707
```

As we can see, the accuracy and the error doesn't vary so much compared to a bigger training set.

Those accuracies were not better enabling the tree to grow deeper, so we deduced that the low results were due to the so few data. Due to that we were not satisfied with the accuracy, we tested the tree with a slightly larger dataset, *segment.txt*, with 7 possible classes, 19 features, and more than 2000 data points. We also set the max_deph constraint. The results obtained were the following ones:



Decision tree segment.txt with 0.3 testing data size

We can see that, in this case, the tree is completely unbalanced, and that (even though is not easily visible in the figure) there are some leaf nodes which have a high uncertainty (big Gini value). The unbalancement is not due to the data, as we also trained the decision tree classifier with *class_weight='balanced'* (for balancing the data in case it was not) and the tree was also equally unbalanced. Despite those nodes, the scoring of this tree is much better than the previous one.

```
Accuracy:  0.8932178932178932
Error:  0.10678210678210676
```

Support Vector Machines

Run SVM to train a classifier, using radial basis as a kernel function. Apply cross-validation to evaluate different combinations of values of the model hyper-parameters (box constraint C and kernel parameter γ). How sensitive is the cross-validation error to changes in C and γ? Choose the combination of C and γ that minimizes the cross-validation error, train the SVM on the entire dataset and report the total classification error.
Advice: consider a binary class problem. Use logarithmic ranges for γ and C.

```
Data used: phishing
Classes: 2
Features: 68
Data-size: 19996
```

We selected this dataset because we thought it would be easier to understand the parametres selection in a binary classification problem.

For the kernel function we the radial basis function: $e^{(-||x-x'||^2)}$

We have selected the following parameters for C and γ.

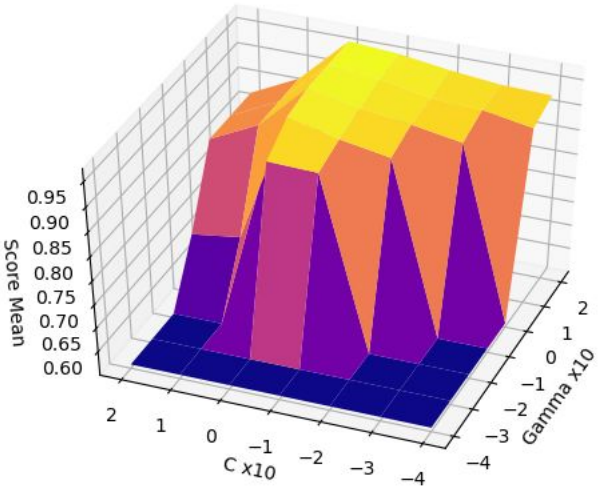
C = [0.0001 0.001 0.01 0.1 1 10 100]
γ = [0.0001 0.001 0.01 0.1 1 10 100]

The C parameter is used to determine the misclassification margin in the training accuracy so that the model can generalize better. We can say that it works like a regularization parameter, such as the alpha in L2 regularization. The lower the value of C the simpler the decision function and the larger the C the best training accuracy but less generalization.

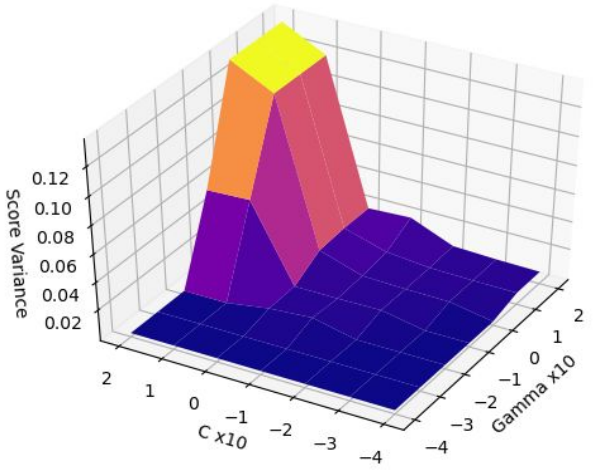
The γ parameter defines how a single point is influenced by the other points of the data set when constructing the margin, it works like a variance. A high value of γ will have only into consideration the data points that are closer to each other to create the boundary and on the contrary, a low gamma will take into consideration the points that are further away. Hence, the higher the gamma, the more restrictive the boundary will be.

And in order to know the effectiveness of each combination of C and γ we use a 5-fold cross validation using the **cross_val_score** function from *scikit-learn*.

These are the results:

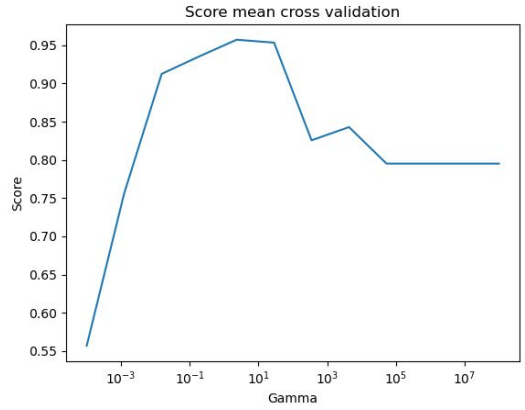


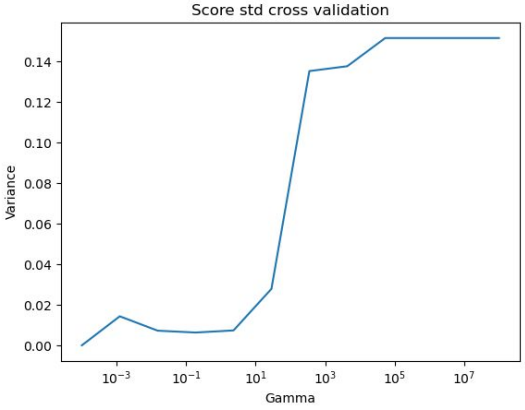
Mean of the cross validation



Variance of the cross validation

As we can see the optimal value for **C** is **1** and the optimal value for is **10** or **100** but we don't how large the value of can be.
If we fix the C parameter to 1 what is the maximum value that can take ?





As we can see for large values of gamma the score does not change, the model remains practically the same. That means that the model can not be more “specific” and being that restrictive means that it does not generalize well and this leads into a good score but not the best we can achieve.

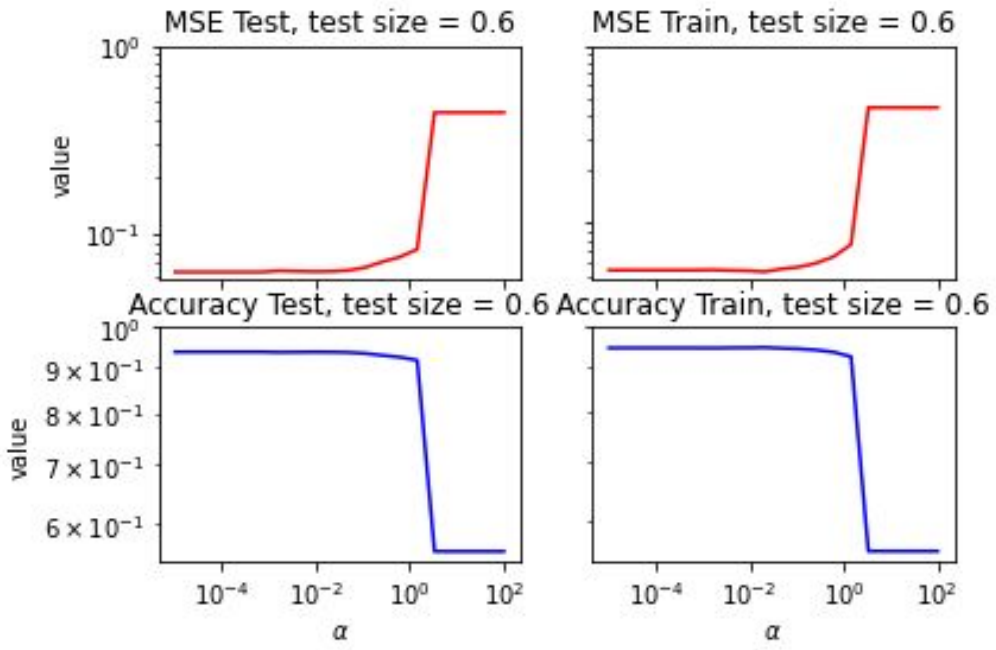
Neural Networks

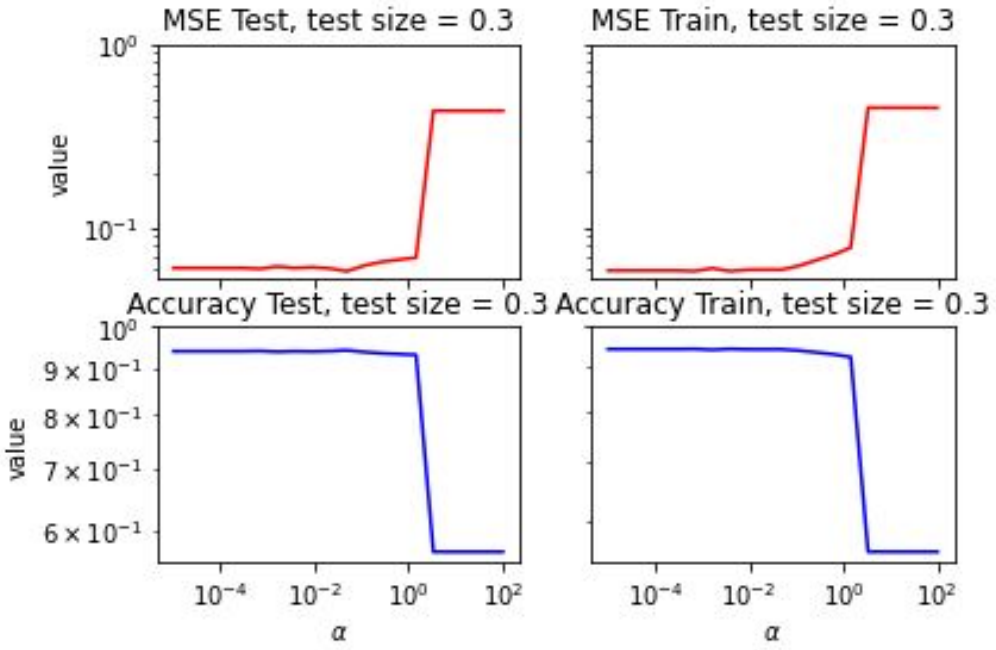
Train a Multi-Layer perceptron using the cross entropy loss with ℓ-2 regularization (weight decay penalty). In other words, the activation function equals the logistic function. Plot curves of the training and validation error as a function of the penalty strength α. How do the curves behave? Explain why.
Advice: use a logarithmic range for hyper-parameter α. Experiment with different sizes of the training/validation sets and different model parameters (network layers).

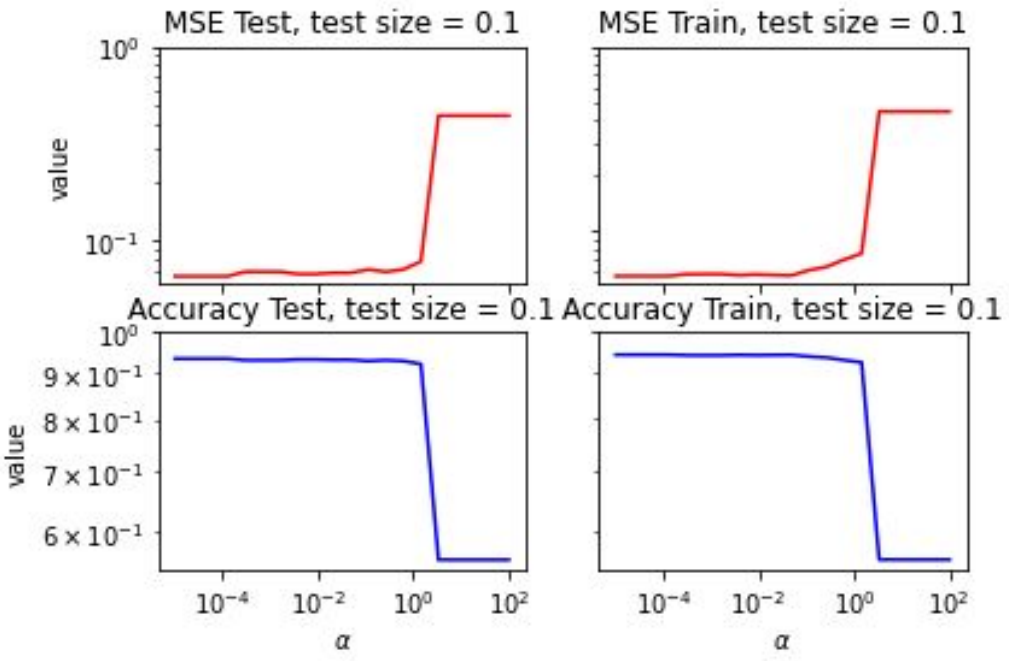
Data used: phishing.txt
Classes: 2
Features: 68
Solver: Adam
Regularization parameter (alpha) = $[1*10^{-5}:1*10^{2}]$ in log space
Hidden layers: 1 layers of 16 neurons [modified for further experiments]

To see how it changes along the alpha and also with different training and testing sets we have computed several permutations, with testing set size from 0.1 to 0.6.

Arnau Colom - U137471
David Moreno - U114148

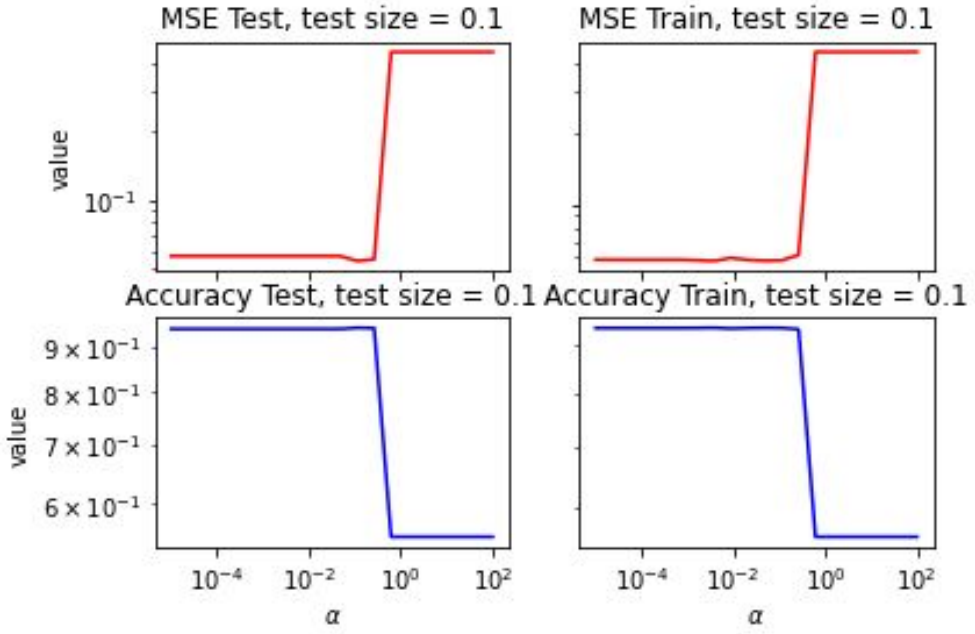






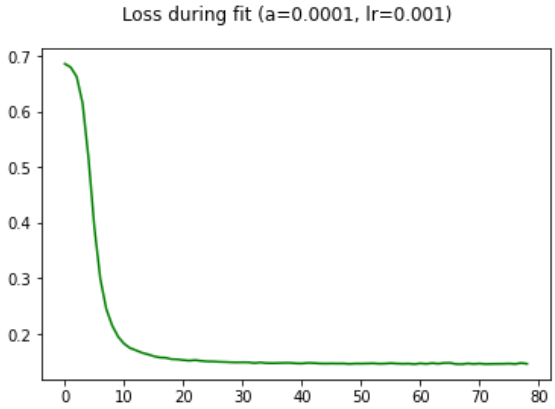
As we can observe, there are no substantial differences between the 0.6 training dataset and the 0.1 one.

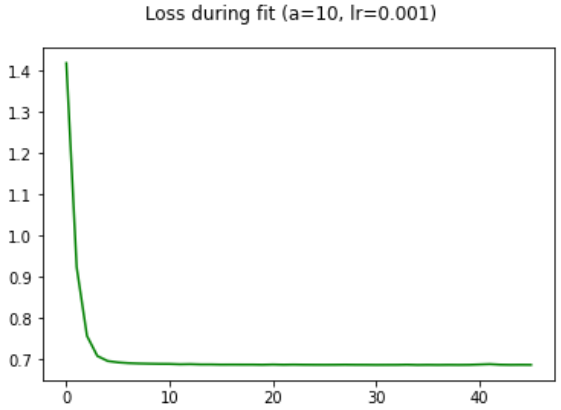
To check that our hidden layer was good enough, we tried to add a second hidden layer, of size 8. As we can see in the figure below, the drop of the accuracy occurs earlier, which mens that with this configuration is more sensible to the regularization parameter.



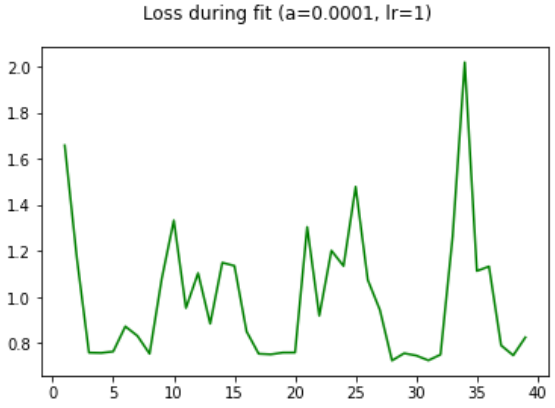
The curves behavior shows that for a small penalization, the model performs pretty well with a ~94% accuracy. Despite that, the decrease of that accuracy, and thus, the increase of the error, is really abrupt. It used to happen when alpha gets closer to 1.

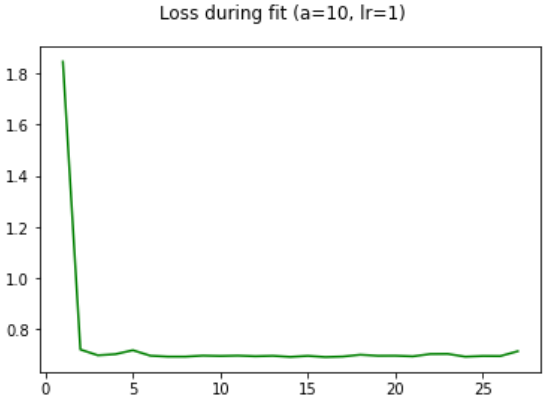
To observe a little bit further on what was going on during the training of the model, we plotted the loss during the fitting for different alphas and learning rates:





As we can observe, with a small value for the penalty value of the regularization (a=0.0001), the loss function is decreasing dramatically as supposed in the first steps. The same happens even when alpha is 10, but with a highest value of the loss (and our accuracy is not good, but the model's loss behaves correctly).





On the other hand, with a bigger learning rate, and a small regularization, we can see an incorrect behaviour of the loss.

In the first case, the model can learn without any restriction, but, since the learning rate is that high, it can not converge correctly. To put a small golf analogy, it would be like hitting the ball with all your force in the putting green, you will be missing the hole (minimum) all the time (only if you are very lucky will you be able to put the ball in the hole).

In the second one what is happening is that the weights can not change, hence the model cannot learn. Following the previous analogy, what is happening is that, when the ball is rolling through the grass, the grass exerts a great friction force. So even if you hit the ball with all your forces it will be a moment that the ball will no longer be able to keep moving forward.