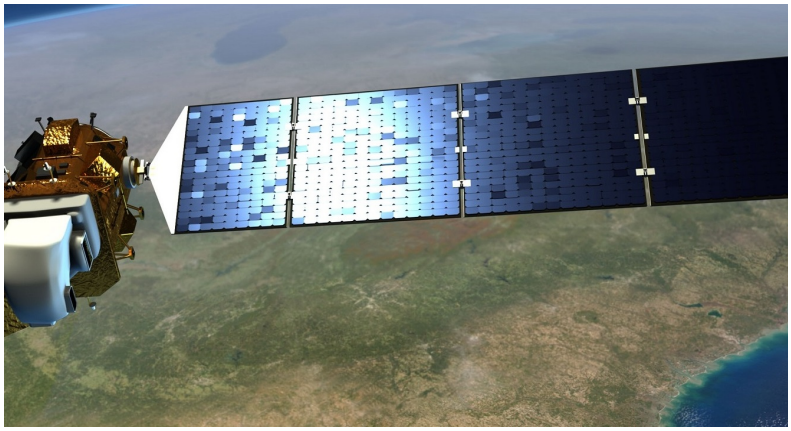
NANYANG TECHNOLOGICAL UNIVERSITY

Durand Tom
N1701729G
15/10/2016

CE4042: Neural Networks

Project 1



Part A: Classification Problem

Statlog (Landsat Satellite) Data Set
https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)

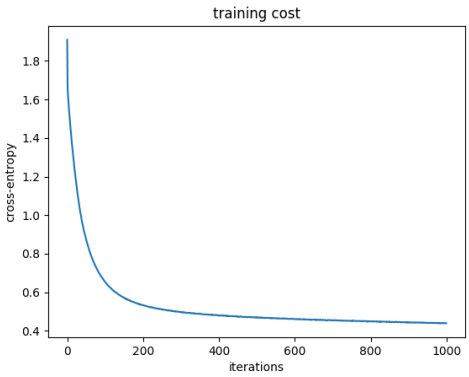
Introduction:

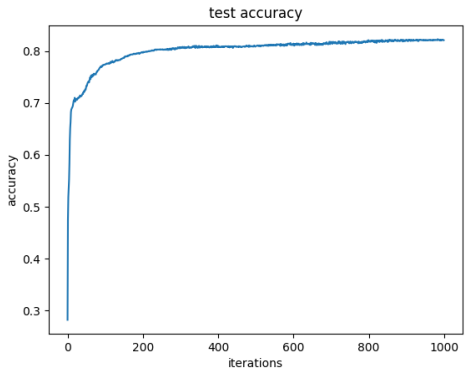
The Landsat satellites are used to observe Earth by capturing a lot of images to exploit them for cartography, climate prediction, agriculture, … A frame captured by those satellites is composed of 4 spectral bands (2 visible, 2 infra-red). Here we want to state the label of each pixel in one region of 82*100 pixels between red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble and very damp grey soil (6 labels). To do so we will use the value of the pixel and its surroundings (3*3 pixels) in the 4 spectral bands. This is typically a problem that can be solved with a multilayer feedforward network for classification. Indeed, these kind of networks has proven its efficiency by solving some difficult problems thanks to the complexity allowed by the adaption of the numerous parameters of the network (not only simple linear boundaries between the labels as for a linear neuron). The use of the softmax activation function for the output layer, as a function of probability, is also very performing for classification.

1 – First design

The first neural network created is a 3-layer feedforward neural network with 10 hidden layer neurons having logistic activation function and the softmax one for the output layer (because we want here to solve a classification problem). The cost function is the cross-entropy, also because we are here in a classification problem. The data is presented as 6435 samples divided into two parts, 4435 for the training and 2000 for the testing. In this part of the project we will use the holdout method to validate our results and estimate the efficiency of our network. Since the data is composed of 37 values ,36 attributes that are the values in the 4 spectral bands for each of the 9 pixels in the neighborhood and the class label (6 possible), the network will have 36 inputs and 6 outputs. We can directly run the code that was provided to begin the project to test a first time the network with as parameters:
-learning rate = 0.01
-decay = 10^{-6}
-batch size = 32
We can notice that the inputs features are already scaled between 0 and 1 to lead to better and more pertinent outputs from the activation function.

This are the results for this network, before any research about the best parameters, parameters that we will try to find later in this project.
Program to run: begin_project_1a.py

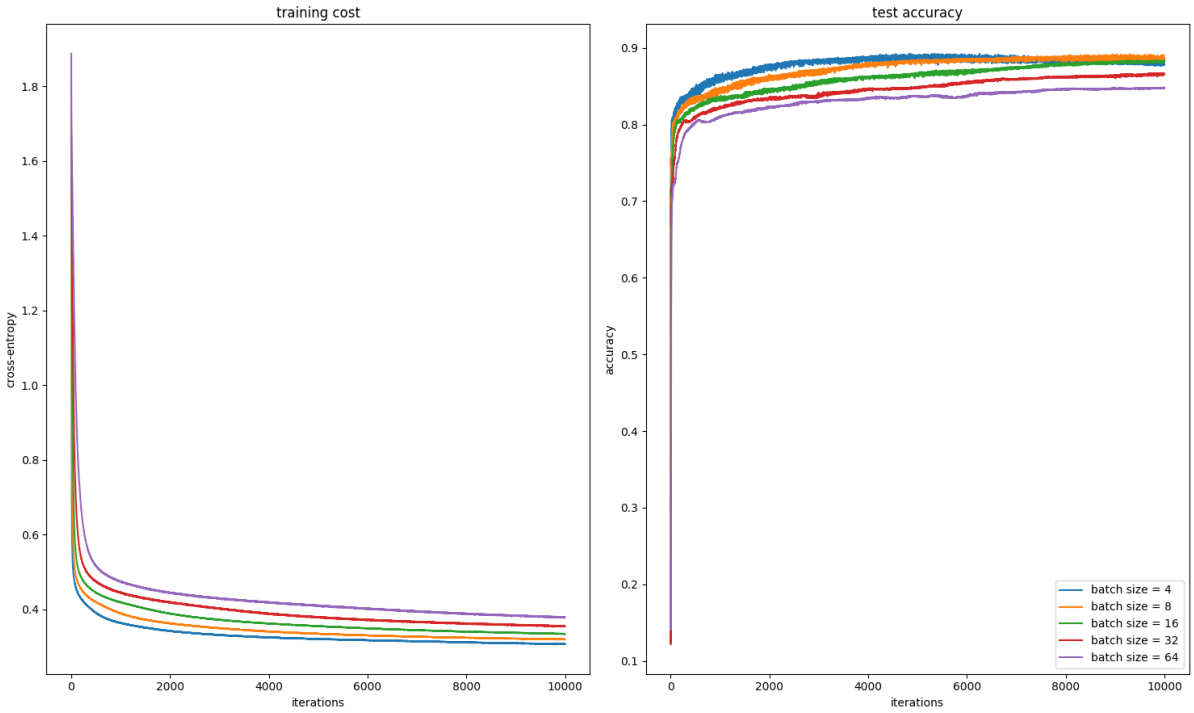




2 – Choice of the optimal batch size.

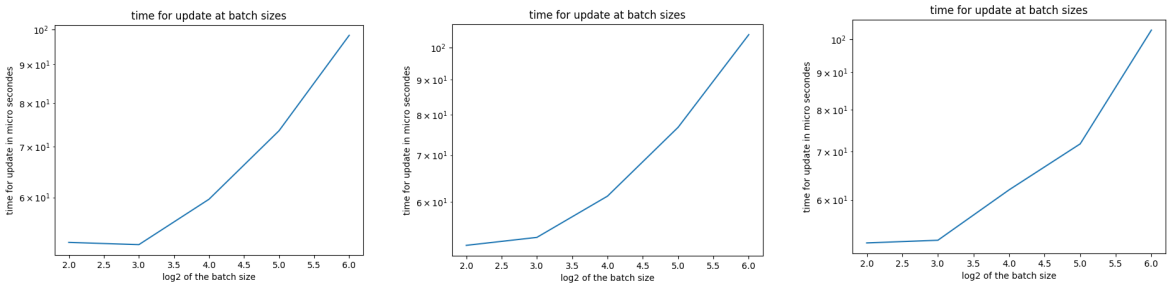
The choice between the speed and smoothness of the convergence with the gradient descent and the accuracy of the stochastic gradient descent method has been resolved by the mini-batch gradient descent technique. Splitting the data into batches allows to have a good balance between performance and rapidity of training. In this part we compared the accuracy (training and testing) and the time to update the parameters for many batch sizes (4, 8, 16, 32, 64).

Program to run: project_A_q2.py



These accuracy's plots show that the convergence of the training curve is faster and better with lower batch size, as we could have guess, and that on the test accuracy the lowest batch size doesn't necessarily lead to the best accuracy, specially if we push the training to

10000 iterations. In fact, we can observe some overfitting for the batch size of four on the training dataset (lower training cost but higher error on the test dataset) and the maximum test accuracy for this batch size is almost the same as for eight and sixteen around 10000 iterations. The best choices presented by those two plots seem to be a batch size of eight or sixteen because they provide a really good accuracy and smoother convergence curves compared to the batch size of 4.



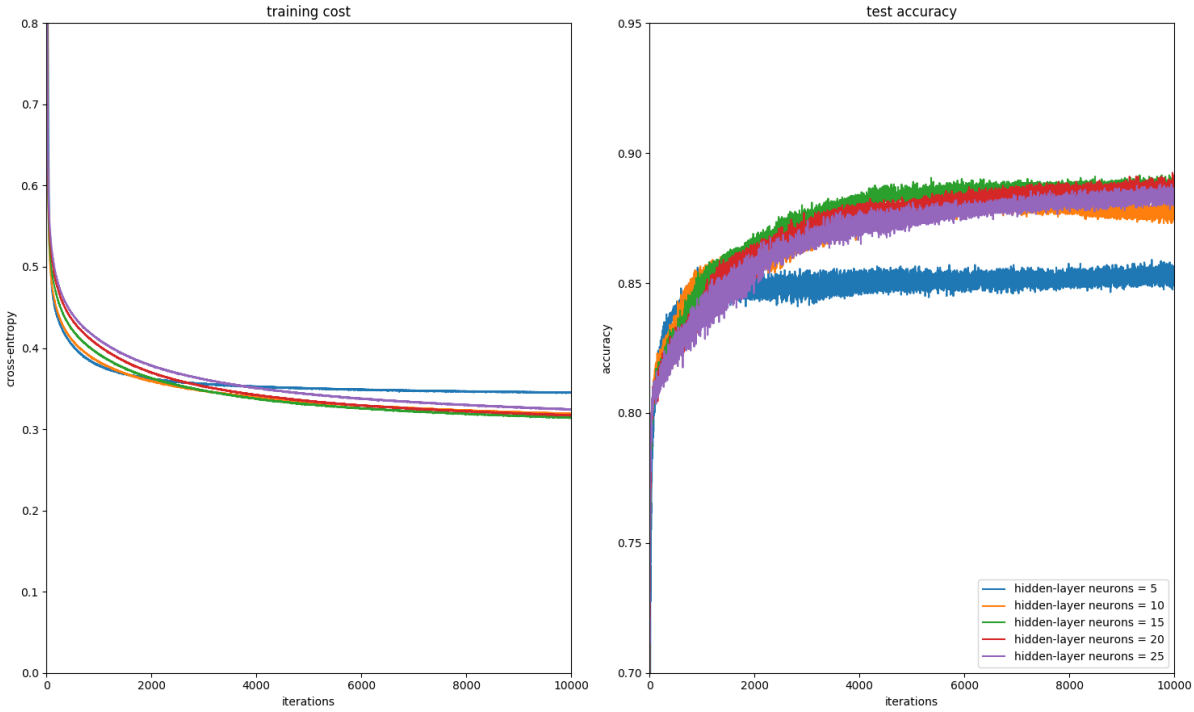
The time for update the weights and biases is also an important point to consider when we choose the batch size for the simple reason that the faster the convergence is reached the more efficient the learning is. The time to update the parameters at batch size is shown above.

Considering the experiments, a batch size of eight is the best choice to do, because it leads to a good accuracy with a quite smooth convergence with a lot of learning iterations per second. We can still admit that a batch size of four with early stopping can also lead to really good performances for the same reasons as before except for the smoothness aspect and that we cannot use a higher learning rate with it.

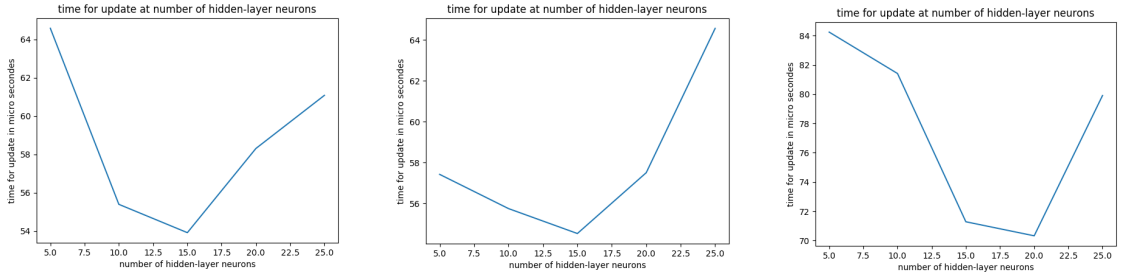
3 – Choice of the optimal number of hidden neurons

Now that we found the optimal batch size we will study the effect of the number of neurons in the hidden layer of the 3 feedforward neural network. Basically a higher number of neurons in the hidden layers of the neurons will provide the possibility to solve difficult problems but it can lead to overfitting with a too good learning of particularities in the training dataset. In the other hand a small number of neuron is not efficient for complicated problems (underfitting) but it can make a better generalization of the scheme learned and then allow good results on the test dataset. To select the optimal number of neuron we need to keep in mind the maximal accuracy on the test dataset and also the time to update the parameters for the same reason as for the batch size.

Program to run: project_A_q3.py



We can see here that the networks with five and ten neurons in the hidden layer are not able to give a really good accuracy. With more neurons the results seem better and we don't observe any overfitting even with the twenty-five neurons network, maybe because we just stopped the training too soon. However, the convergence is a bit faster with fifteen hidden neurons, this is normal because we need less training, iterations, to find the best weights and biases if there are fewer of them.



Those curves represent three different results obtained during the training of the network. The first thing that we can see is this U-form of the curves that suggests that there is an optimal number of hidden neuron to have fast updates of the parameters. We can also see that they are not exactly the same and that the time for update varies for each of them but it is directly due to my computer and its activity during the running of the program.

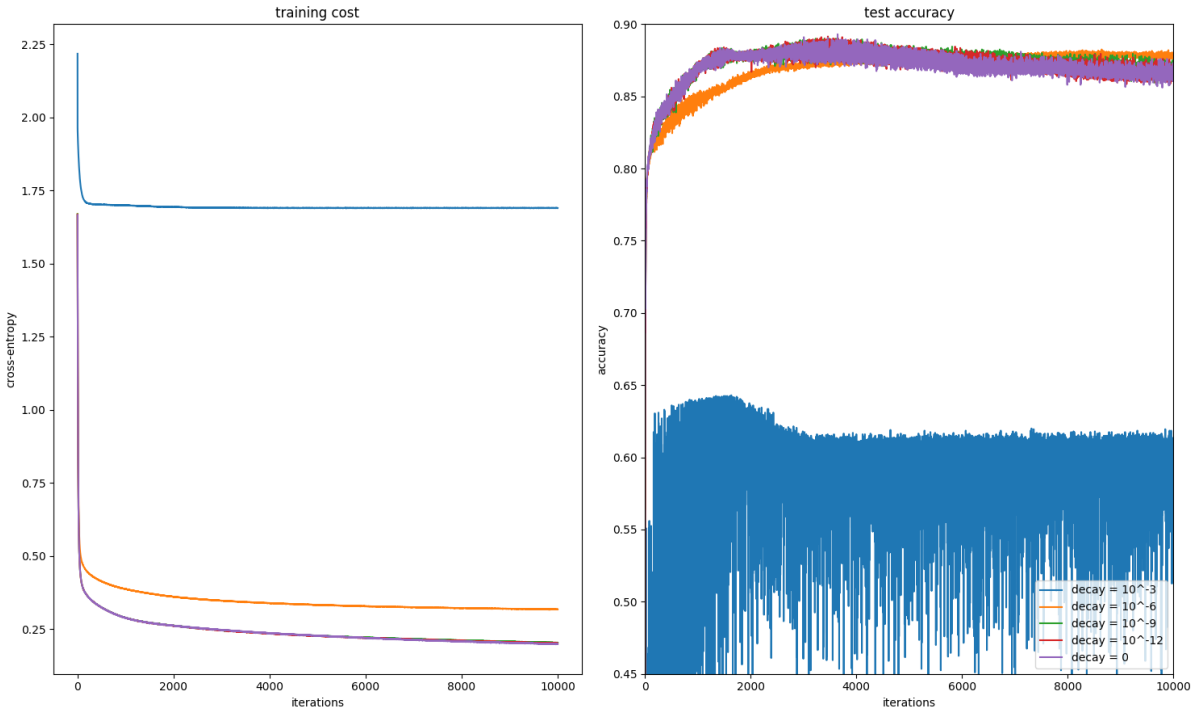
Thanks to the time to update and the accuracy plots we can easily say that fifteen hidden neurons is the optimal choice. At 10000 iterations it seems to be a good way to do the classification and more neurons doesn't seem to lead to a way better convergence if we

continue the training, and it is also, by taking the mean of my results, the optimal number to update the parameters as fast as possible.

4 – Choice of the optimal decay parameter

A problem that usually appears during training neural networks is overfitting. To avoid this many techniques are possible and one of them is the regularization of the weights. The principle is to add a penalty term in the cost function that corresponds to the sum of the square of the weights. We do this in order to avoid them to be too big, thing that usually happens when overfitting and that leads to a bad capability of the network to generalize what he learned and so to have good results on the test dataset. This term is multiplied by a parameter called the decay that will make him more or less important.

Program to run: project_A_q4.py



As we can see on the figure, with a decay too big (here 10^{-3} for example) the network really struggles to learn because the cost function is to big and the changes in the weights are to important at each iteration so it can't converge. For a decay of 10^{-6} we can see that the test accuracy curve converges and is really stable, the accuracy does not go as down as we can observe for the smaller decay parameters. It really avoids the overffiting of the network. If we just want to use this method to avoid overfitting then 10^{-6} is the optimal value for the decay but we can see that the accuracy and also the training cost cannot go to really good values. Better results can be obtained with early stopping for the smallest values of the decay (around

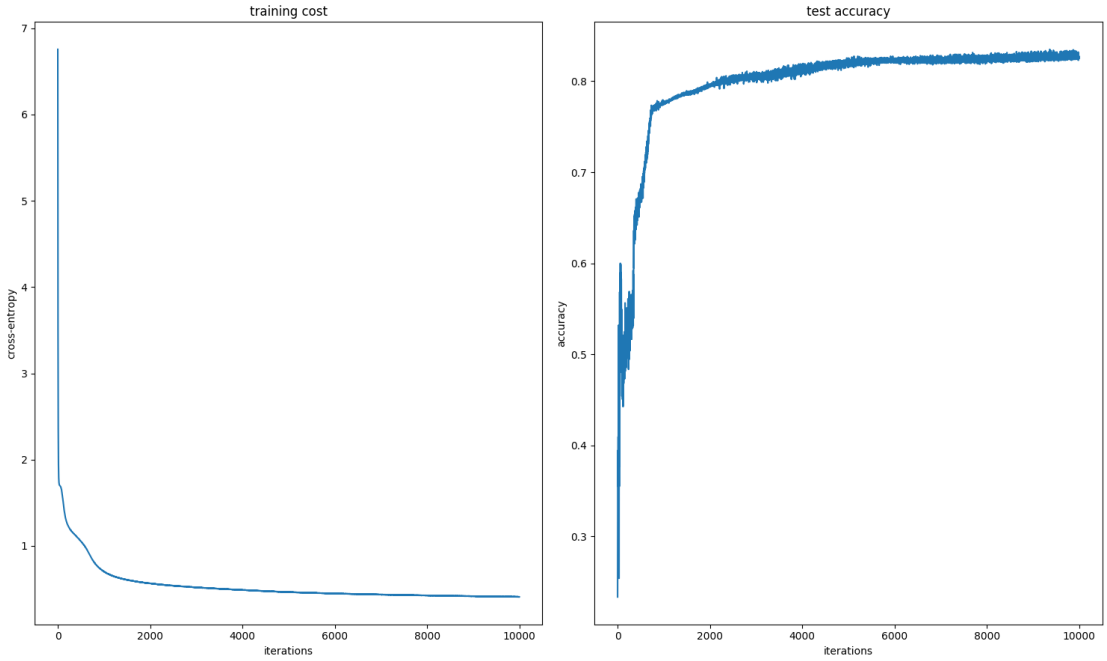
3500 iterations) that, by the way, provide all almost the same results. This is why this result of 10^{-6} must be discussed and maybe a decay is not really needed in this classification problem. In fact, the training dataset is maybe really close to the test dataset and it is not needful to use a weight regularization.

5 – Performance of a 4-layer network

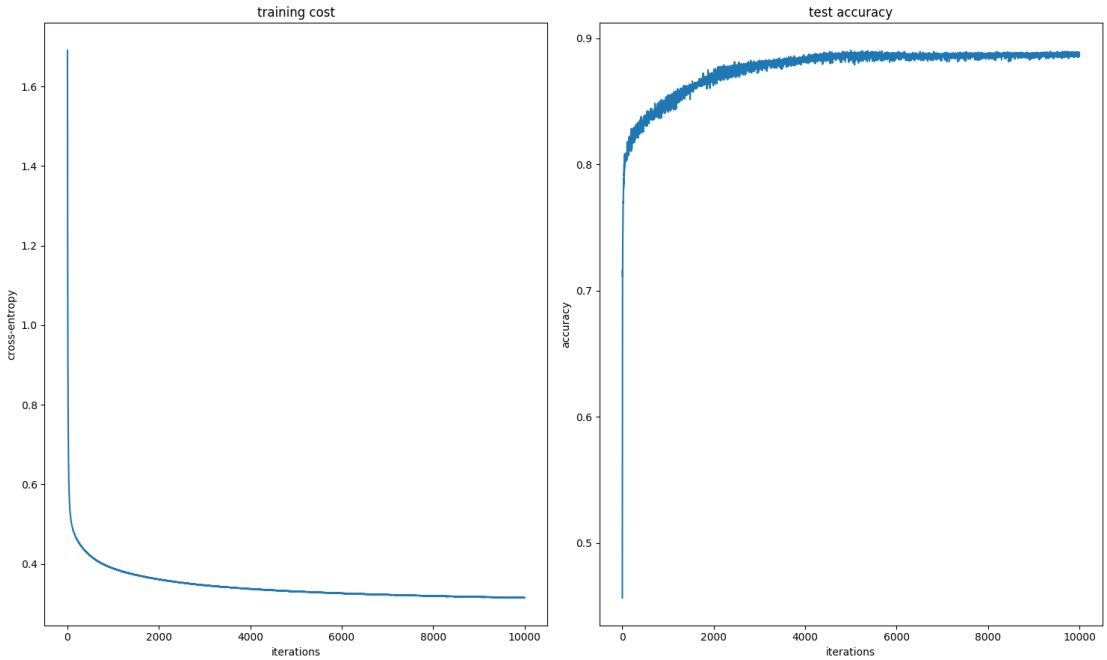
In this part we will compare the performances of a 4-layer network with 2 hidden layers of 10 neurons (logistic activation function), a decay of 10^{-6} and a batch size of 32 and our 3-layer network. As we said, a higher number of neurons or a deeper network with many layers can perform better for difficult problem, classification with a lot of parameters that can be trained.

Program to run: project_A_q5.py and project_A_final_3_layer.py

To compare the two networks, we will just use the accuracy obtained of the training data and the test accuracy even if there is a lot of other things to compare to state the advantages and drawbacks of them like the time to update for example. The first figure is about the 4-layer network and the second one about the 3-layer network with the optimal parameters stated before.



Performances of the 4-layer network above and 3-layer under.



The first thing we can observe is that the convergence is not very clean and smooth for the testing accuracy of the deeper neural network. This is normal because it needs more training to find the best weights and biases since they are more. Comparing the two networks' curves we can see that the 3 layers converges to better performances for the training and the testing accuracy: 89% accuracy against 83.5% for the 4-layer stopping at 10 000 iterations (the precision for the 4-layer does not really increase after, it arrives to a maximum of 86% at 47 000 iterations but it is really noisy). To conclude, the convergence is smoother, faster and has better performance with the 3-layers network even if we pushed the training iterations for the 4-layers (it could have maybe just needed more training).

Finally, in this first part of the project we tried to solve a classification problem with a neural network approach. The principle was to find the best parameters for the network, using some different metrics to compare the performances with the holdout method for example. We can notice that the result is quite convincing since the testing accuracy increased of almost 8% since the first network and we also minimized the time to update. We can also see that for some problems we don't need to use a too deep or complicated neural network, sometimes a simpler one can perform better, as we saw in part 5. An other point is that when we cannot really represent the data, when the dimension of the inputs is quite big, the only way to design the best network is to try the different parameters and compare the performances.

Part B: Approximation Problem

California Housing
http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

Introduction:

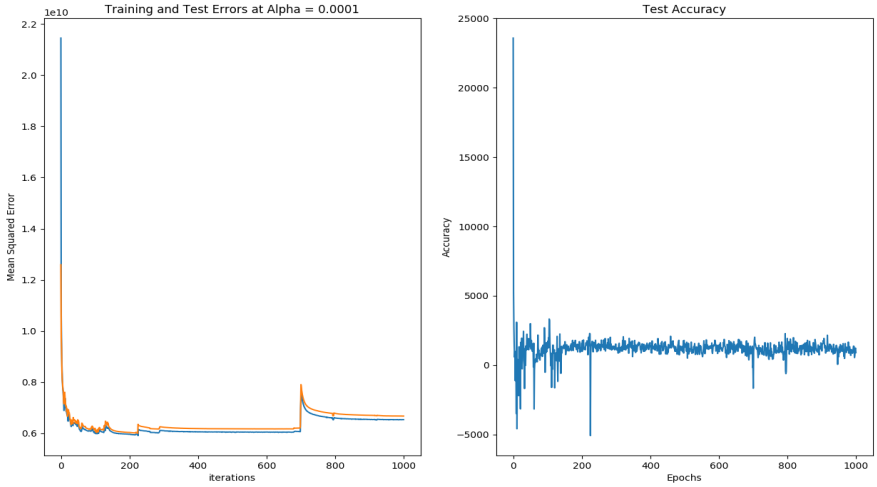
The price of housing has always been a complicated task to solve but the issues are really big in this domain, for the sellers but also for the buyers that want to invest in a new neighborhood for instance. In this part we will try to approximate the price of housing complex in California with a multilayer feedforward network. What is interesting in the use of neural network and especially multilayer network is that it will learn by itself how much each of the specs of houses are important so we don't need an intermediate person to evaluate as best as her can the impact of the number of bedrooms or the size of the house on the price. Speaking about the specs, the inputs features are the median income, the housing median age, the total number of rooms, of bedrooms, the population, the households, the latitude and longitude as the distance to the centroids of each block. The data comes from all the blocks group in California since 1990. An other point is that the multilayer network will be also able to see if there are some correlations between some inputs and exploit them to provide an accurate result (thing that is really hard to see for us, specially on a big dataset as the housing in California).

1 – First design

The first neural network created for this second problem is a 3-layer feedforward neural network with 30 hidden layer neurons having logistic activation function and, because we only want to predict the price of the house, a linear neuron for the output neuron. A linear neuron allows us to skip the mapping of the output activation function. The error that we will try to minimize is the mean squared error because it is the most relevant one in function approximation. The data is presented as 20 640 samples and each sample consists of 8 inputs data concerning the housing complexes (longitude, latitude, housing median age, total rooms, total bedrooms, population, households and the median incomes) and the target is the median house value. To evaluate the performances of our models we will use the 5-fold cross validation on the train data and we split the total data into 30% for testing and 70% for training. This means that for all experiments we will choose the optimal parameter regarding to the 5 results obtained with the validation datasets for each value of the parameter and evaluate the final accuracy on the test dataset.

For the first network these are the results we obtained, training the network on all the training dataset and testing it on the test dataset with the following parameters:
-learning rate = 0.0001
-hidden neurons = 30
-batch size = 32

Program to run: begin_project_1b.py



We can see that the network converges even if one pick occurs around 7000 epochs and the error one the test accuracy is of about 2000 dollars that is already quite reasonable compared to the price of a house.

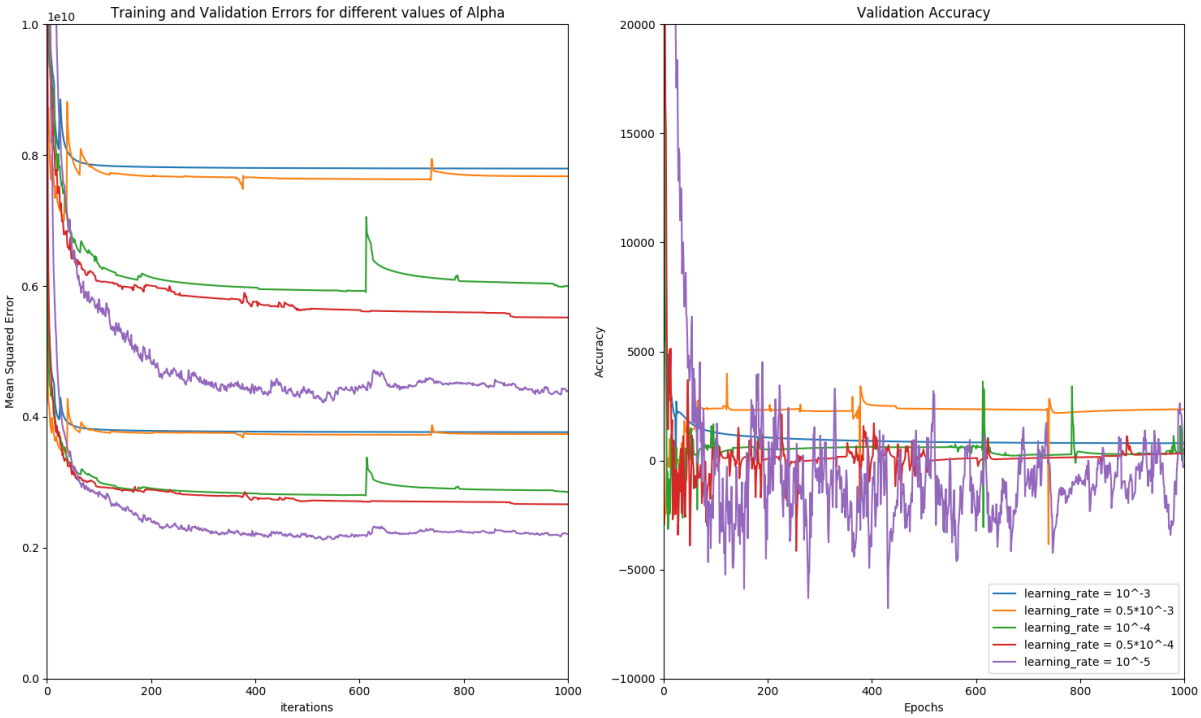
2 – Choice of the optimal learning rate.

The choice of the learning rate allows us to control the updates of the parameters of the networks because we will choose the importance of the gradient in the equation. By putting a higher learning rate we will have a faster convergence since the modification of the weights will be more important at each update but it can also lead to a bad accuracy because the network will not be able to find the optimal small variations of the weights to have the best performances.

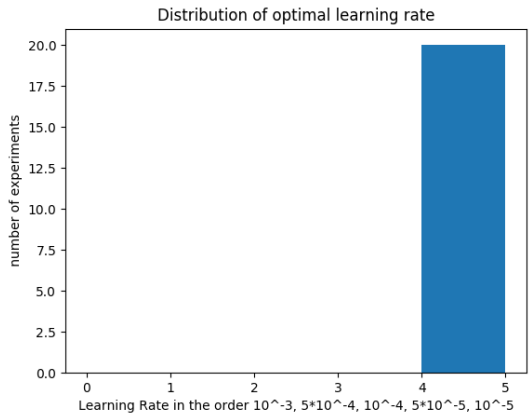
In our case we tried many different learning rates and used the average minimal error with a 5-cross validation test on 20 experiments to see which learning rate allows the best accuracy. The minimal average error is the value we use to evaluate the network for each training because it is really easy to remember what was the best iteration, version of the network so it is logical to consider just the minimal error. The fact that we compute the mean of this one along the 5 fold gives even more justification to its use.

Program to run: project_B_q2.py

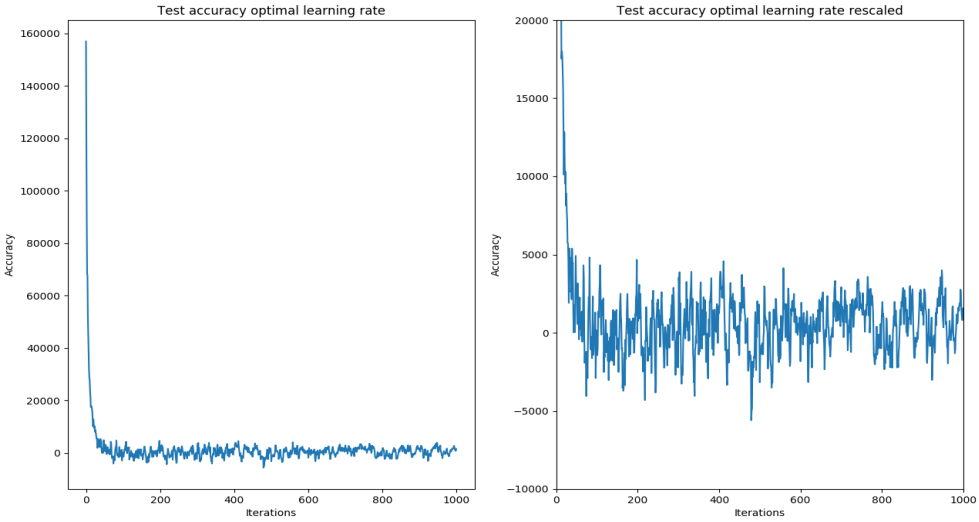
NB: For the left plot the validation error is always the curve above the training error for each value of the learning parameter.



We can see that a higher learning rate provides a smoother (but with big jumps sometimes, maybe due to a too small batch size) and faster convergence for the mean squared error but this one stays quite high compared to a lower learning rate that can achieve way better accuracy and a smaller error. The best solution would be to choose a high learning rate at the beginning and reduce it as we advance in the iterations of training in order to benefit from the faster convergence, at least at the beginning, and then the good accuracy, even if noisy here (normally a smaller learning rate will provide a smoother convergence), of a smaller learning rate. But we have to choose only one value and for this we will use 20 experiments with the 5-fold cross validation and the minimal error as stated before.



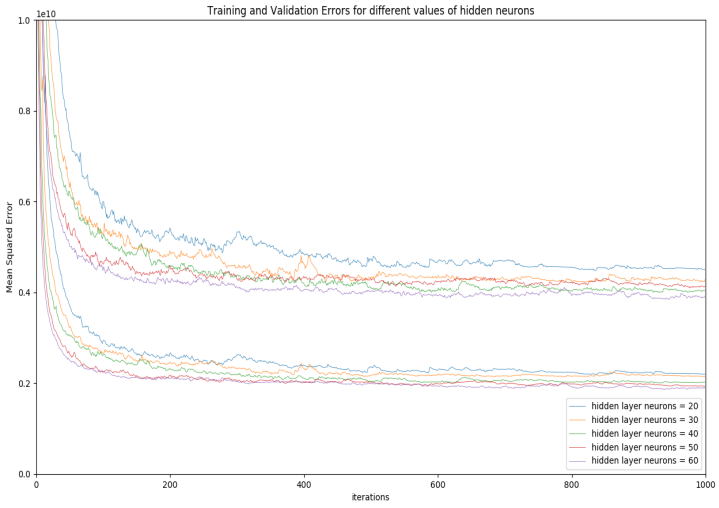
Regarding to the histogram we can see that for all the experiments the network with the learning rate of 10^-5 had the best minimal error and this is why we will use this value for the next parts of the project. The following plot is the test accuracy on the unstudied test dataset after training of all the training dataset (training and validation) with different scales.



This learning rate achieves a good accuracy on the test dataset, specially compared to the first experiment. The noise is not that important since the mean of the accuracy seems to be around 0, and the curves is still smooth if we compare this noise to the price of a house.

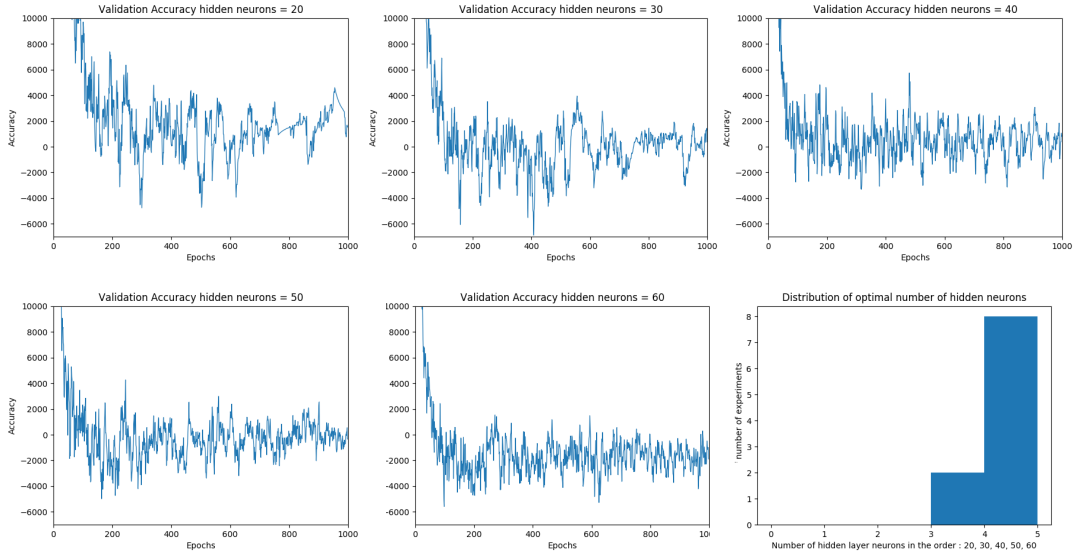
3 – Choice of the optimal number of hidden neurons.

We already state what was the impact of the number of neurons in the hidden layer during the part A so we will go through this part a bit more quickly. What we are interested about by using the cross validation on this project is to select each time the parameter that will provide us the best possible accuracy. This is why we will not use the time to update as a standard to select the number of hidden neuron for this question.

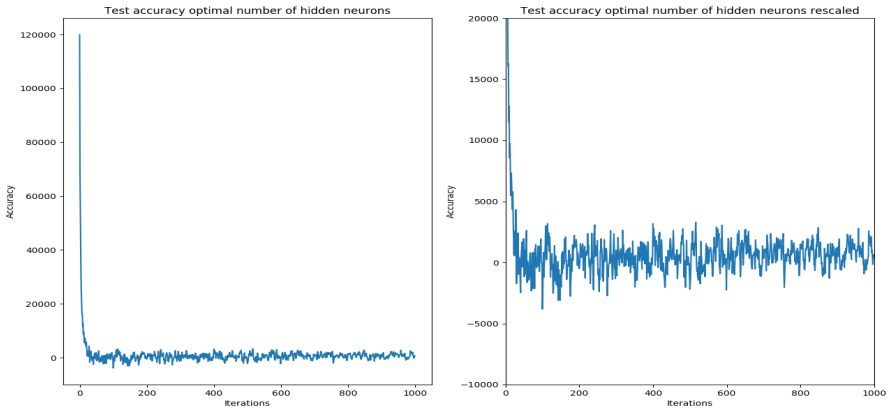


Program to run: project_B_q3.py

NB: The validation error is always the higher curve for each value of the learning rate parameter.

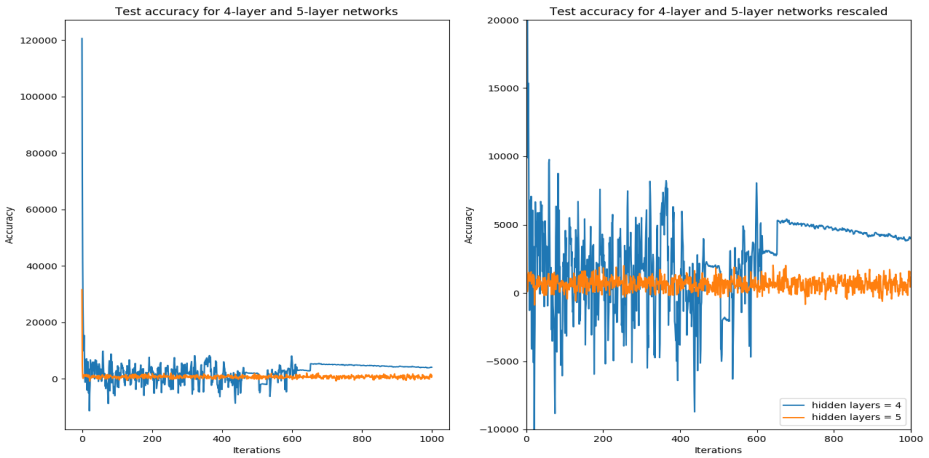


On the training and validation errors (first figure) we can see that the 60-neurons network performs the best, specially on the validation error, followed by the 40-neurons. The accuracy's plots are hard to exploit because all the curves are quite similar. The thing we can notice is a high number of neuron provides a less noisy curve so that can be a sign that the network needs it to map correctly the function. An other point why we cannot really exploit those curves is that they represent just one fold during one experiment for each setting and this is why we use the last histogram to fix the number of neurons. By referring to the average minimal error with the cross validation on 10 experiments we can see that 60 is the optimal setting and we will continue the project with this value for the 3-layer network. This is the test accuracy we obtain with it (we can see that the curves are better that in the step 2):



4 – Performances of deeper neural networks, 4 and 5 layers

We saw before that a large number of neurons was needed to perform better on this problem. This is a legitimate question to ask ourselves if a deeper neural network will perform even better since the function to map seems complicated. This is what we will study in this part a 4 and 5 layer networks compared to the 3 layers' model. To do so we will plot the test errors for the new networks to see their performances and we will compare them with the plots of part 3.



The accuracy's curves obtained show that a 5-layer network performs way better the 4-layer one. It is interesting because the convergence of the deeper network is way faster than a 4 or 3 layers network even if there are more parameters to find and update. But after few more experiments I admitted that it was the case for this dataset. These are the minimum errors and the best accuracies you obtain with the 3, 4 and 5-layers network (obtained by running the programs) on the test step:

```
3-layers network : Minimum error: 3919812109.2, Best accuracy 1145.4,
Number of Iterations: 865
4-layers network : Minimum error: 11499146618.4, Best accuracy 540.9,
Number of Iterations: 585
5-layers network : Minimum error: 13556414213.8, Best accuracy -0.3,
Number of Iterations: 853
```

By comparing all of the test accuracy curves and those numbers we can see that the best one is the 5-layer. The 4-layers network suffers from a lot of noise during training but has a way better best accuracy and minimal error than our tuned 3-layers network but it is questionable since in general the 3 layer is way more stable so maybe on an other test dataset it would be more advised to use the 3-layers network. Since those testing curves are always converging on this range of epochs we can also state that no overfitting occurs so we could have continued a bit more the training.

To conclude on this second part, we tried to solve a function approximation with a multilayer perceptron. We searched for the optimal settings that will give us the best possible accuracy, using the 5-folds cross validation to select them. We can see that this method was really useful since the curves are not always really easy to exploit (many noise, experiments, scale, …) and gave us stable results among all the experiments. On the last part we also saw that sometimes even when we try to tuned a network to perform the best he can a deeper network can just outperform it easily, without any particular setting configuration.

NOTES:

Computer specs:

-MacBook Pro (Retina, 13-inch, Mid 2014)
-Processor 2,6 GHz Intel Core i5
-Memory 8 GB 1600 MHz DDR3
-Graphics Intel Iris 1536 MB
-Startup Disk Macintosh HD 128 GB
-Python 2.7.13

Images sources:
Landsat
https://spaceflightnow.com/2015/04/21/multiple-satellites-planned-in-long-term-landsat-program/
Houses
http://www.terrafirmaglobalpartners.com/real-estate-market-news/housing-affordability-at-forefront-of-ca-realtor-concerns/