INF8225 TP3 - Rapport de laboratoire

Apprentissage dans un réseau de neurones

Maxime Schmitt
1719088

15 mars 2016

1 2 hidden layers Neural Network optimization

for each *theta* in ($\theta_{firstHiddenLayer}$,$\theta_{secondHiddenLayer}$, $\theta_{outputLayer}$) **do**
$theta[:, :-1] \leftarrow rand()$ ▷ W (without the bias)
$theta[:, -1] \leftarrow 0$ ▷ B (only the bias)
end for

while loss function didn't converge **do**
Segmentation of the input and output training data in minibatches *miniBatchesInput* and *miniBatchesOutput*
for each (X,Y) in (miniBatchesInput, miniBatchesOutput) **do**
 # Feed forward
Adding a line of 1 at the end of X (bias)
$activation_{firstHiddenLayer} \leftarrow activationFunction(\theta_{firstHiddenLayer} * X)$
Adding a line of 1 at the end of X $activation_{firstHiddenLayer}$ (bias)
$activation_{secondHiddenLayer} \leftarrow activationFunction(\theta_{secondHiddenLayer} * activation_{firstHiddenLayer})$

Adding a line of 1 at the end of X $activation_{secondHiddenLayer}$ (bias)
$activation_{outputLayer} \leftarrow outputActivationFunction(\theta_{outputLayer} * activation_{secondHiddenLayer})$

 # Back propagation
$\delta_{outputLayer} \leftarrow -(Y - activation_outputLayer)$
$gradientW_{outputLayer} \leftarrow \delta_{outputLayer} * activation_{secondHiddenLayer}[:, :-1]$
$gradientB_{outputLayer} \leftarrow \delta_{outputLayer} * activation_{secondHiddenLayer}[:, -1]$

$\delta_{secondHiddenLayer} \leftarrow derivatedActivationFunction(activation_{firstHiddenLayer}) * \theta_{outputLayer} * \delta_{outputLayer}$
$gradientW_{secondHiddenLayer} = \delta_{secondHiddenLayer} * activation_{firstHiddenLayer}[:, :-1]$
$gradientB_{secondHiddenLayer} = \delta_{secondHiddenLayer} * activation_{firstHiddenLayer}[:, -1]$

$\delta_{secondHiddenLayer} = derivatedActivationFunction(X) * \theta_{secondHiddenLayer} * \delta_{secondHiddenLayer}$
$gradientW_{firstHiddenLayer} = \delta_{firstHiddenLayer} * X[:, :-1]$
$gradientB_{firstHiddenLayer} = \delta_{firstHiddenLayer} * X[:, -1]$

 # Parameters update
$\theta_{firstHiddenLayer}[:, :-1] = \theta_{firstHiddenLayer}[:, :-1] - learningRate * gradientW_{firstHiddenLayer}$
$\theta_{firstHiddenLayer}[:, -1] = \theta_{firstHiddenLayer}[:, -1] - learningRate * gradientB_{firstHiddenLayer}$

$\theta_{secondHiddenLayer}[:, :-1] = \theta_{secondHiddenLayer}[:, :-1] - learningRate * gradientW_{secondHiddenLayer}$
$\theta_{secondHiddenLayer}[:, -1] = \theta_{secondHiddenLayer}[:, -1] - learningRate * gradientB_{secondHiddenLayer}$

$\theta_{outputLayer}[:, :-1] = \theta_{outputLayer}[:, :-1] - learningRate * gradientW_{outputLayer}$
$\theta_{outputLayer}[:, -1] = \theta_outputLayer[:, -1] - learningRate * gradientB_{outputLayer}$
end for
end while

2 Experiments and hyper parameters optimization

2.1 Considered parameters

The hyper parameters that we can change are :

— **batchNumbers** : Number of mini batches to use

— **learningRate** : Learning rate of the network

— **activationParameter** : Value of the coefficient for the piecewise linear activation function

— **WInitValue** : Maximal initial value for the random initialization of θ

— **convergenceThreshold** : value that determines when we consider that the loss function has converged

— **hiddenLayersNumber** : Number of hidden layers in the network

— **firstHiddenLayerSize** : Number of nodes in the first hidden layer

— **secondHiddenLayerSize** : Number of nodes in the second hidden layer (when applicable)

Experimentally, we found that the following values for these parameters made a good combination :

— **batchNumbers** = 5000

— **learningRate** = 0.01

— **activationParameter** = 0.01

— **WInitValue** : 0.1

— **convergenceThreshold** : 0.01

— **hiddenLayersNumber** : 1 or 2

— **firstHiddenLayerSize** : 300

— **secondHiddenLayerSize** : 100 (when applicable)

Note that this set doesn't necessarily provide the best result but it provides a good result while not requiring too much time to run and was chosen considering these two critera.

2.2 Comparison of one and two hidden layer networks

In this section, we provide the results and the learning curves for the networks using the parameters given previously, the two only differing from each other with the number of hidden layers they contain :

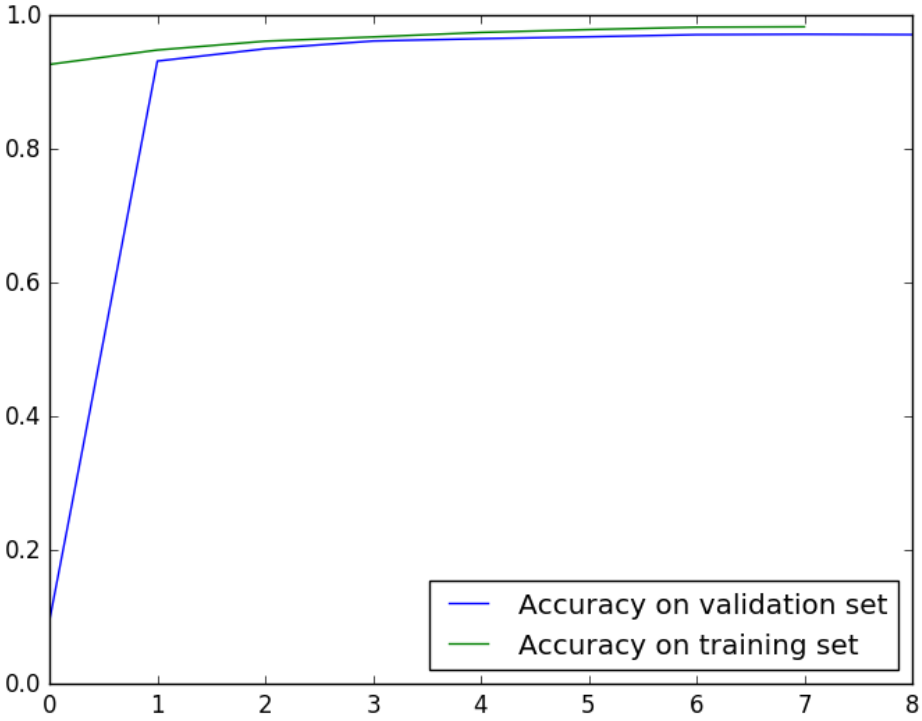
— The first network contains only one hidden layer of 300 nodes

— The second network contains two hidden layers : the first one of 300 nodes and the second one of 100 nodes

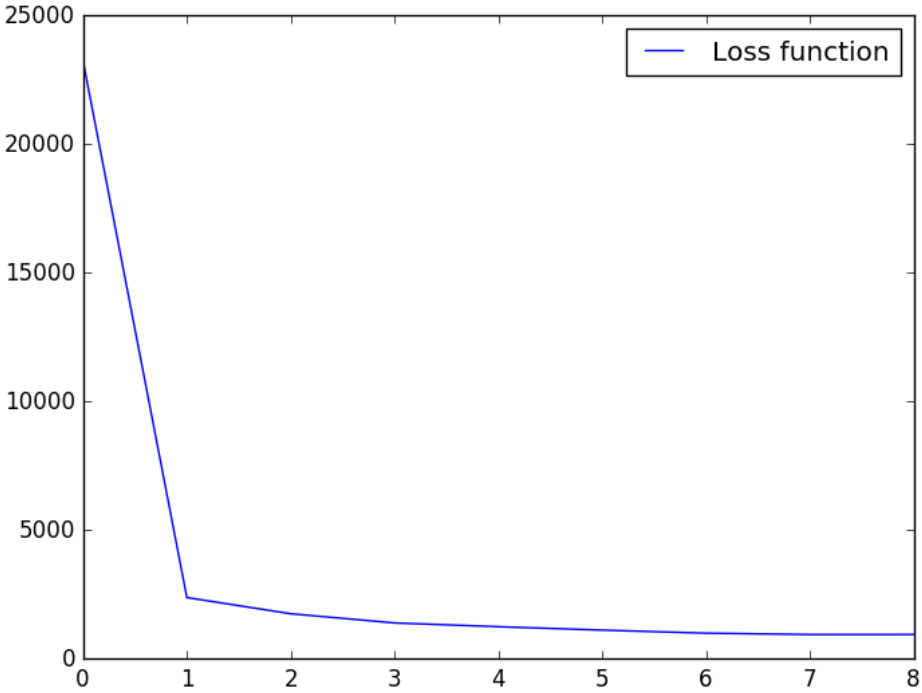
On the figure 1 and figure 2 we can see respectively the learning curves of the network with 2 hidden layers and with 1 hidden layer.

At the end we also have the following accuracies on the testing :
97.03% for the network with two hidden layers 97.4% for the network with one hidden layer

The least deep seems to provide a better resutl, but we can imagine that with some finer parameters tuning we could be able to get better results with the deepest one, especially considering that our current parameters converge in only 8 iterations.

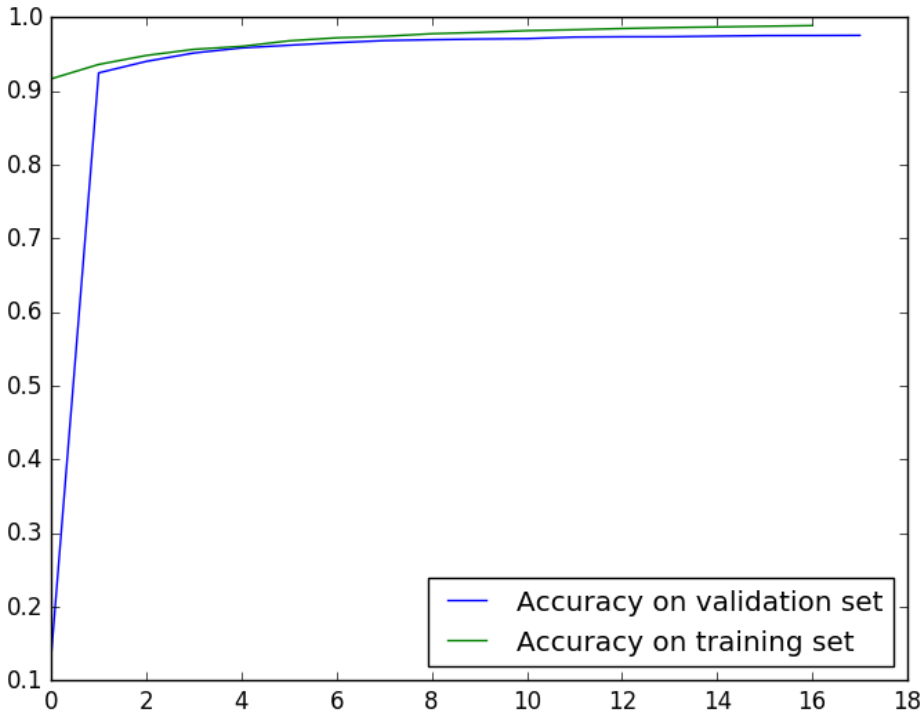


(a) Accuracy on the training and the validation sets

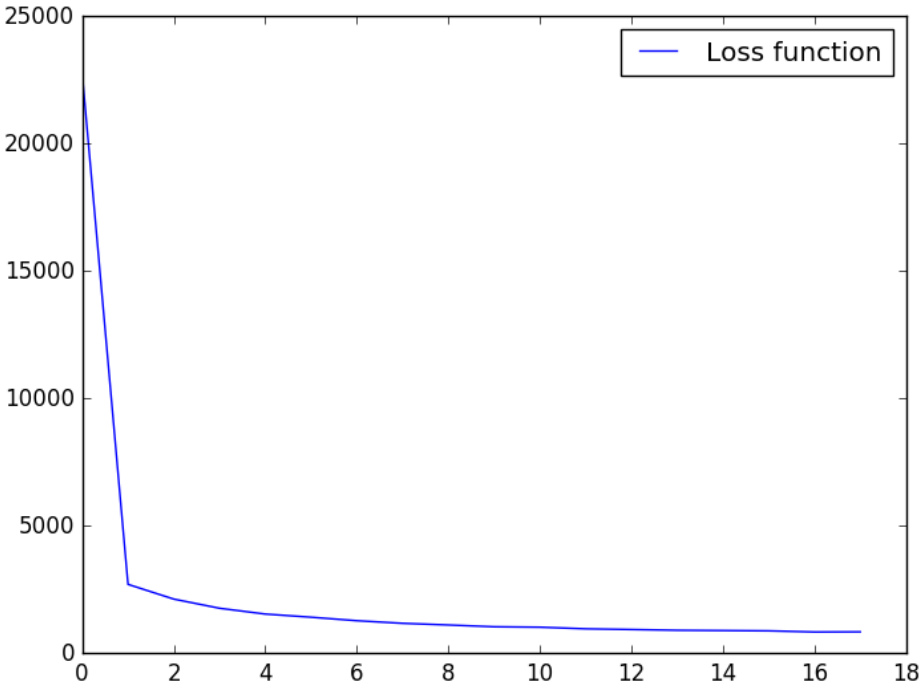


(b) Loss function

FIGURE 1 – Curves for the network with two hidden layers



(a) Accuracy on the training and the validation sets



(b) Loss function

FIGURE 2 – Curves for the network with one hidden layer

3 Conclusion

With this work, we succeeded in implementing a multi-layer neural network and made some experiment to get some intuition on the influence of the various parameters on the result of the learning, and compared two networks of different deepness. However this comparison is quite simplistic and a more thourough study should be done to obtain a clear result.

Moreover, the proposed implementation could be improved by adding a regularization term to the learning, to avoid some overfitting and therefore get better results on the testing data set at the end of the learning phase.