Backpropagation

Nicolas Brissonneau , UTEID:nb24488 , email:nicolasb@utexas.edu

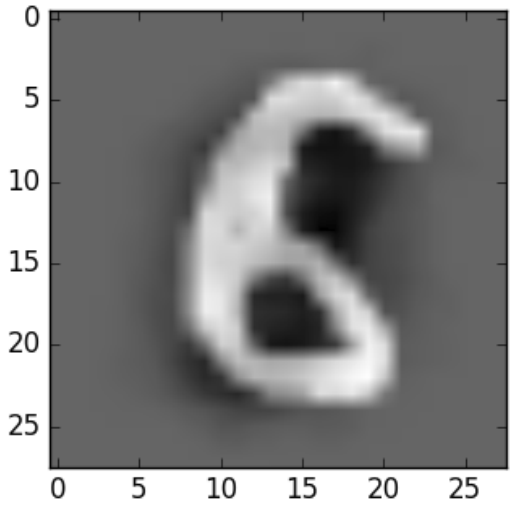

Fig. 1. Wrong hyper-parameters bounds

I. Introduction

I am considering the problem of identifying a "testing sample" S_{test} composed of n_{test} handwritten digits similar to Fig.1, each originally encoded as a 28x28 gray-scaled picture (where brightness corresponds to the pixel value) which will be named V_{im} and of dimensions 784x1, we will denote $V_{im}(j)$ the j-indexed image from the samples list.

II. Method

We are setting up a neural network which is made of 3 layers:

- Input layer: Receiving a transformed V_{im} (784x1)
- Hidden layer: Intermediate layer (n_{hidden}x1)
- Output layer: Returning the classified digit (10x1)

By naming the neurons returned values from the first layer to the last x_0, x_1 and x_2, we can express our objective function:

$$E = \frac{1}{2}\|x_2 - d\|^2 \tag{1}$$

Where d is the desired output. We need to compute x_2 and x_1, we do so using the following forward propagation equation:

$$x_{k+1} = g(W_k x_k) \tag{2}$$

With W_k the k_{th} matrix of weights between the k_{th} and $k_{th}+1$ layers of neurons and g being the sigmoid function:

$$g(u) = \frac{1}{1+e^{-u}} \tag{3}$$

Now we want to backpropagate the error through the network to update the weights according to the objective function. We know the hamiltonian matrix H from the Euler-Lagrange formulation:

$$H = \frac{1}{2}\|x_k - d\|^2 + \Sigma_{k=0}^{k=K-1}(\lambda_{k+1}^T[-x_{k+1} + g(W_k x_k)] \tag{4}$$

K being associated with the last layer, now differentiating H with respect to x_k:

$$H_{x_k} = -\lambda_k + [W_k^T \Delta_{k+1} g'(W_k x_k)] \tag{5}$$

Where Δ is the square matrix of diagonal λ_i and g' is the differentiated sigmoid:

$$g'(u) = g(u)(1 - g(u)) \tag{6}$$

Now we can compute all λ_k with:

$$\begin{aligned} \lambda_K &= x_K - d \text{ for k=K} \\ \lambda_k &= -W_k^T \Delta_{k+1} g'(W_k x_k) \text{ for k=0,..,K-1} \end{aligned} \tag{7}$$

And finally compute the weight adjustment:

$$\frac{\partial H}{\partial w_{k_{i,j}}} = \lambda_{k+1_i} x_{kj} g'(w_{ki} x_k) \tag{8}$$

III. Results

First of all, we verify our algorithm is working correctly by making sure the error is reduced properly by backpropagation over a number of iterations. To do so, we set up a training scenario in which we measure the norm of the error described in the objective function: $error = x_2 - d$. For the following tests we are using N=100 neurons in the hidden layer, 5000 sample images per epoch and a learning rate of 0.1. In Fig. 2 we are testing the classifier's performance over the data samples it has been trained on. One can see that the mean of the norms of the errors decreases over each epoch.

In the following test shown in Fig. 3 we used a similar setting as for Fig. 2 but this time we are interested into testing the classifier's performance on a data set it has not been trained on before. Out of fairness with the testing demonstrated on the training set, we test this test the neural network on 5000 sample images from the testing set. We notice a drop, less significant than when performing on the training data.

Now one big question which was asked during this study is which number of neurons to choose for the hidden layer. To answer this question I ran two series of tests, the first one focusing on a number of neurons inferior to 100 and the second one exploring higher numbers of neurons. In Fig. 4 we notice a strong improvement of performance from 1 to

Fig. 2. Evolution of the error norm over epochs while classifying the training data

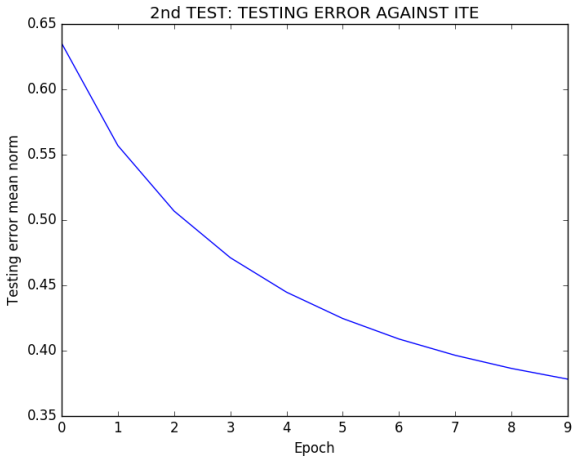

Fig. 3. Evolution of the error norm over epochs while classifying the testing data

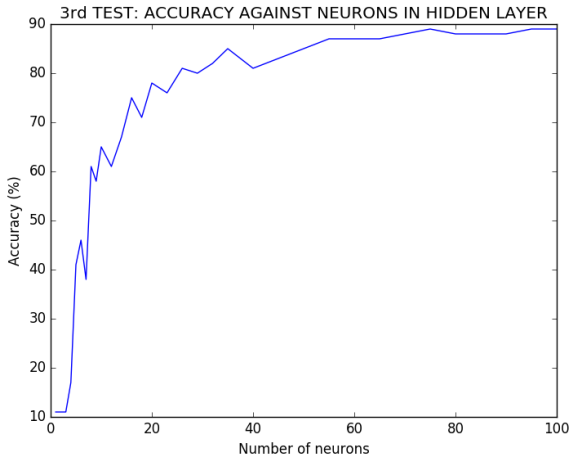

Fig. 4. Accuracy for low number of neurons

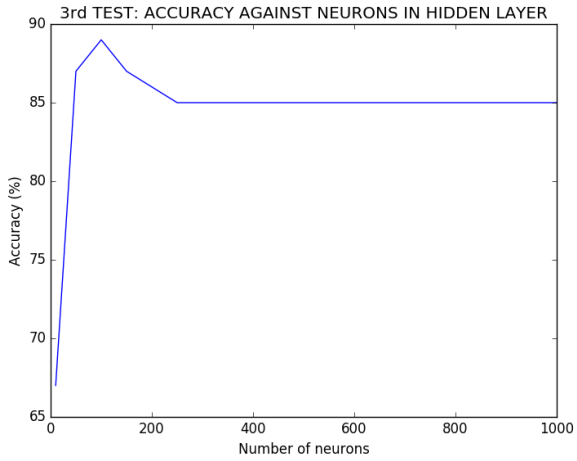

Fig. 5. Accuracy for high number of neurons

30 neurons, and a slow but steady increase until 100. In Fig. 5 however, we notice a slight decrease that past 100 neurons followed by a null slope which infers little chance to notice a change in performance beyond these values. We will now consider the changes when adding another hidden layer. In this case scenario we have two hidden layers, each of size 200x1. What we can observe in Fig. 6 and Fig. 7 is a much slower convergence, it seems that adding a layer, for a similar configuration and neural network parameters, will slow the learning. Though intuitively speaking, more layers should allow us to grasp more complex patterns and increase the accuracy on the long run, it proves to be a much slower process.

IV. Conclusion

I learned that neural networks and backpropagation require meticulous tuning. I noticed that including a bias term helped a lot with the convergence of my neural nets, and I learned it was possible to derive by hand a different version of the algorithm shown in class. We have also tried to use semi-linear activations but it did not converge.

Future works include randomly deactivation neurons to stimulate a better learning, trying different activation functions and testing different hidden layers configurations.

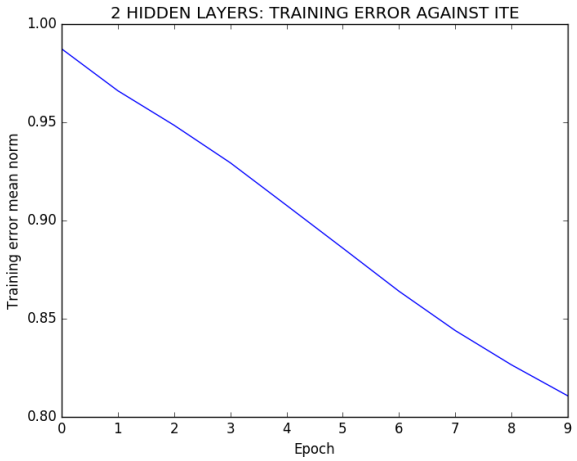

Fig. 6. Evolution of the error norm over epochs while classifying the training data for 2 hidden layers

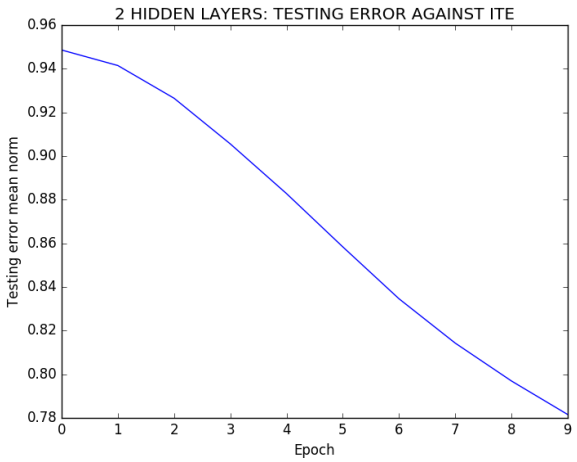

Fig. 7. Evolution of the error norm over epochs while classifying the testing data for 2 hidden layers