INF554: Machine Learning I
École Polytechnique
Assignment

Neural Networks Learning for hand-written digit recognition

C. Giatsidis, S. Limnios, J. Read, N. Tziortziotis and M. Vazirgiannis

October 2017

1 Description of the Assignment

The objective of the specific project is to implement a general purpose Neural Network and to apply it to the task of hand-written recognition. For the purposes of this project, we will use the MNIST database [1] that consists of handwritten digit images $(0-9)$. Actually, it is divided in $60,000$ examples for the training set and $10,000$ examples for testing.

All digit images have been size-normalized and centered in a fixed size image of 28×28 pixels. Each pixel of the image is represented by a value in the range of $[0, 255]$, where 0 corresponds to black, 255 to white and anything in between is a different shade of grey. In our case, the pixels are normalized and represent the features of our dataset; therefore, each image (instance) has 784 features. That way, the training set has dimensions $60,000 \times 784$ and the test set $10,000 \times 784$. Regarding the class labels, each figure (digit) belongs to the category that this digit represents (e.g., digit 2 belongs to category 2).

2 Neural Networks - Pipeline

In this task, you should examine and compare the performance of Neural Networks with different number of hidden layers and units per layer, in the problem of hand-digit recognition. For notation simplicity, let us consider the neural network illustrated in Figure 1. It consists of 3 layers: an input layer, a hidden layer and an output layer. Moreover, let x be the input of our network and $\theta = (\Theta_1, \Theta_2)$ represent the network's parameters.

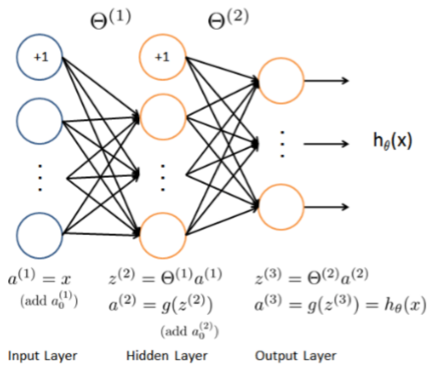

Figure 1: Neural Network with one hidden layer

[1]The MNIST database: `http://yann.lecun.com/exdb/mnist/`.

Useful note: Please take into account that your implementation should be general. More specifically, it should work for any number of hidden layers and any number of units per layer.

2.1 Loading the MNIST dataset

Firstly, we load the MNIST dataset by calling the function `read_dataset.py`. This returns the variables `images_training` and `labels_training` that contain the training instances and their class labels, respectively. In a similar way, the test data and their class are loaded in the variables `images_test` and `labels_test`, respectively. Each pixel of the image is represented by a floating point number in the range of $[0, 1]$ indicating the grayscale intensity at that location. The pixels of each image are "unrolled" into a 784-d vector. In this way, each one of these examples becomes a single row in our data matrices.
Useful note: In order to keep the complexity reasonable, you can use a lower number of instances for training. This can be achieved by changing the value of the variable `size_training` which is located in the `main.py` script.

2.2 Setting up Neural Network Structure

In the second part, you asked to set up the structure of the NN. Actually, you will be asked to select both the number of hidden layers as well as the number of nodes for each one of them.

In the continue, you have to randomly initialize the neural network's parameters. An effective way is to select random values for the parameters uniformly in the range $[-\epsilon, \epsilon]$, where ϵ is equal to a small value (e.g., $\epsilon = 0.12$).

Tasks to be done

- Initialize the NN parameters in the `randInitializeWeights.py` file.
- In the final report, please explain (the reason) why it is important to randomly initialize the NN's parameters. What happens in the case where we set equal to zero all neural network's parameters? Finally, you should examine the sensitivity of your model as regards the ϵ parameter.

2.3 Sigmoid Function

In the third part, you should implement the sigmoid function given as:

$$g(z) = \frac{1}{1 + \exp(-z)} \tag{1}$$

Tasks to be done

- Implement the sigmoid function in the `sigmoid.py` file.

Useful note: When $z = 0$, the sigmoid should be exactly equal to 0.5.

2.4 Gradient of the Sigmoid Function

In the fourth part, you should implement the gradient of the sigmoid function given as:

$$g'(z) = \frac{d}{dz} g(z) = g(z) \odot (1 - g(z)). \tag{2}$$

Tasks to be done

- Implement the gradient of the sigmoid function in the `sigmoidGradient.py` file.

Useful note: For large absolute values of $|z|$, the gradient should be close to 0. When $z = 0$, the gradient should be exactly equal to 0.25.

2.5 FeedForward and Cost Function

In the fifth part, you should implement the cost function of the neural network, which computes the cost according to the following formula:

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right], \tag{3}$$

where K is equal to the total number of possible labels (10 in our case) and m is the total number of training instances. Note that $h_\theta(x^{(i)})_k$ represents the activation of the k^{th} output unit.

Tasks to be done

- Implement the cost function of the neural network in the `costFunction.py` file. Actually, you need to implement the feedforward computation that calculates the $h_\theta(x^{(i)})_k$ for every instance i and sum the cost over all instances.

Useful note: You need to recode the image labels as vectors containing only values 0 or 1. For instance, if the corresponding label of instance $x^{(i)}$ is equal to 3, then $y^{(i)}$ should be a 10-dimensional vector with $y_3^{(i)} = 1$, and all the other elements equal to 0. Moreover, after implementing the cost function, the `main.py` script will call the `checkNNCost.py` script that examines if your code computes the cost correctly.

2.6 FeedForward and Cost Function with Regularization

In the sixth part, you should implement the cost function of the neural network considering the regularization factor. The cost function with regularization is given as:

$$\begin{aligned}J(\Theta) &= \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]\\ &+ \frac{\lambda}{2m}\left[\sum_{j=1}^{40}\sum_{k=1}^{784}\left(\Theta_{j,k}^{(1)}\right)^2 + \sum_{j=1}^{10}\sum_{k=1}^{40}\left(\Theta_{j,k}^{(2)}\right)^2\right],\end{aligned} \tag{4}$$

where K is equal to the number of possible labels ($K = 10$ in our case) and m corresponds to the total number of training instances. In the aforementioned formula, we assumed that the hidden layer composed by 40 nodes.

Tasks to be done

- Fill the `costFunction.py` function by considering the cost for the regularized terms.

Useful note: You should not regularize the terms that correspond to the bias factors. Moreover, after implementing the cost function, the `main.py` script will execute the `checkNNCost.py` script. Thus, you will have the opportunity to examine if your code computing the cost correctly.

2.7 Backpropagation

In this part, you should implement the backpropagation that calculates the gradient of the cost function. Once you have computed the gradient, the gradient is fed to an advanced optimization method which in turn uses it to update the weights (stochastic gradient could be also used). In the following, we present the basic steps of the backpropagation algorithm.

Initialization: Set $\Delta_{i,j}^{(l)} = 0, \forall i, j, l$.

You should implement steps 1 to 4 in a loop considering one instance (sample) at a time.

1. Set the input layer's values $a^{(1)}$ to the the t^{th} training example $x^{(t)}$. Perform a feedforward pass by computing the activations $(z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)})$

2. For each output unit k in the outer layer (layer 3 in our case, see Fig.1), we set
$$\delta_k^{(3)} = \left(a_k^{(3)} - y_k\right) \tag{5}$$
3. For the hidden layer, set
$$\delta^{(2)} = \Theta^{(2)^\top}\delta^{(3)} \odot g'(z^{(2)}) \tag{6}$$
4. Accumulate the gradient from this example using the following formula. Note that you should skip or remove $\delta_0^{(2)}$ that corresponds to the bias factors.
$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} a^{(l)^\top} \tag{7}$$
5. Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $\frac{1}{m}$:
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)} \tag{8}$$

Tasks to be done

- Implement the backpropagation algorithm in the `backwards.py` file.

Useful note: After implementing the backpropagation algorithm, the `main.py` script will execute the `checkNNGradient.py` script. Therefore, you will have the opportunity to examine if your code computing the gradients correctly.

2.8 Backpropagation with regularization

Now, you should consider the reguralization factor to the gradient. Actually, you can add this as an additional term after computing the gradients using backpropagation, as:
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)} \quad \text{for } j = 0 \tag{9}$$
and
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} \quad \text{for } j \geq 1 \tag{10}$$

Tasks to be done

- Fill the `backwards.py` script by adding the reguralization term to the gradient computed at the previous step.

Useful note: Please keep on your mind that you do not need to regularize the first column of $\Theta^{(l)}$ which is used for the bias term. After implementing the backpropagation algorithm, the `main.py` script will call the `checkNNGradient.py` script, with the regularization parameter set to 3. Thus, you will have the opportunity to examine if your code computing the gradients correctly.

2.9 Training Neural Network

After you have successfully implemented the neural network cost function and gradient computation, the next step is to use an advanced optimization for training the neural network in order to discover a good set of parameters. For simplicity purposes, we consider the optimization process as a "black box" (the L-BFGS-B algorithm used for optimazation) (**feel free to examine different optimization schemes**).

Tasks to be done:

- Fill the `predict.py` script which takes as input a number of testing instances (`images_test`) and returns a vector of their predicted labels.

- You have to train and examine a number of different neural network schemes (different number of hidden layers and units per layer).
- You have to train and examine the above neural network schemes for more iterations (e.g., set MaxFun to 400) and also vary the regularization parameter λ.
- **Bonus task:** Consider different activation functions on the hidden layers. For instance, use the hyperbolic tangent function instead of the sigmoid function in our example.

3 Details about the Submission of the Project

Your final evaluation for the project will be based on both the accuracy that will be achieved on the test dataset, as well as on your total approach to the problem.

- A 1-2 pages report, in which you should present and explain your results (e.g. accuracy).
- A directory with the code of your implementation.
- Create a .zip file with name `firstname_lastname.zip`, containing the code and the report and submit it to moodle.