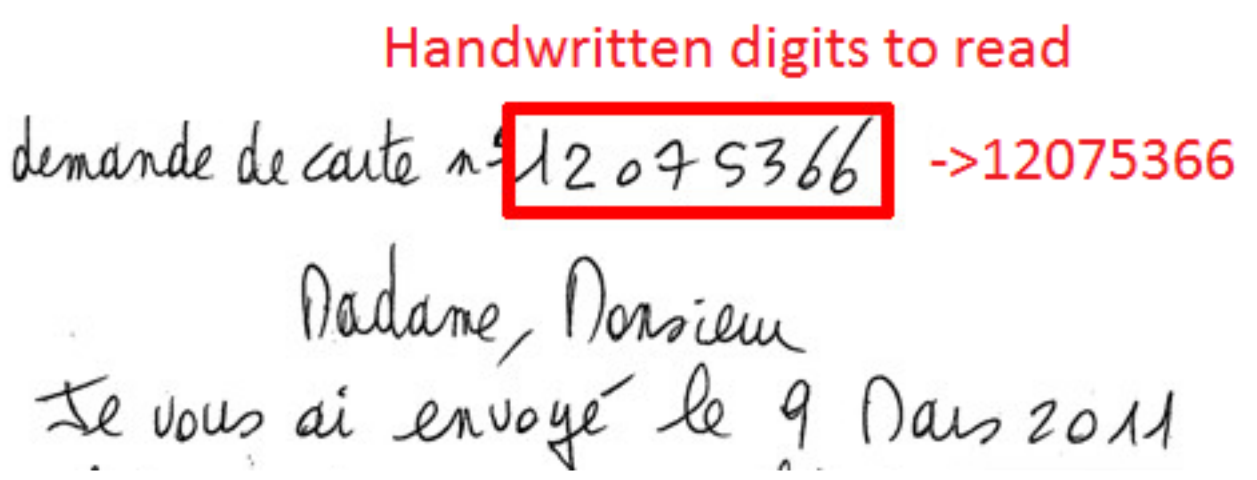
PART 1 - OPTION B : Handwritten Digit Recognition with Neural Networks

Image by Olivier Augereau


Handwritten digits to read
demande de carte n° 12075366
->12075366
Madame, Monsieur
Je vous ai envoyé le 9 Mars 2011

For this project, you will build and train two systems for digit classification: one neural network and one linear classifier. Note: for the classifiers and settings suggested in this project, overfitting is not a significant problem; we therefore forego requiring that you have a validation set. You can nevertheless use one at your discretion.

The Input

You will work with the MNIST dataset. The dataset is available in easy-to-read-in format here, with the digits already seprated into a training and a test set. I recommend you divide the data by 255.0 (note: the .0 is important) so that it's in the range 0..1.

About code

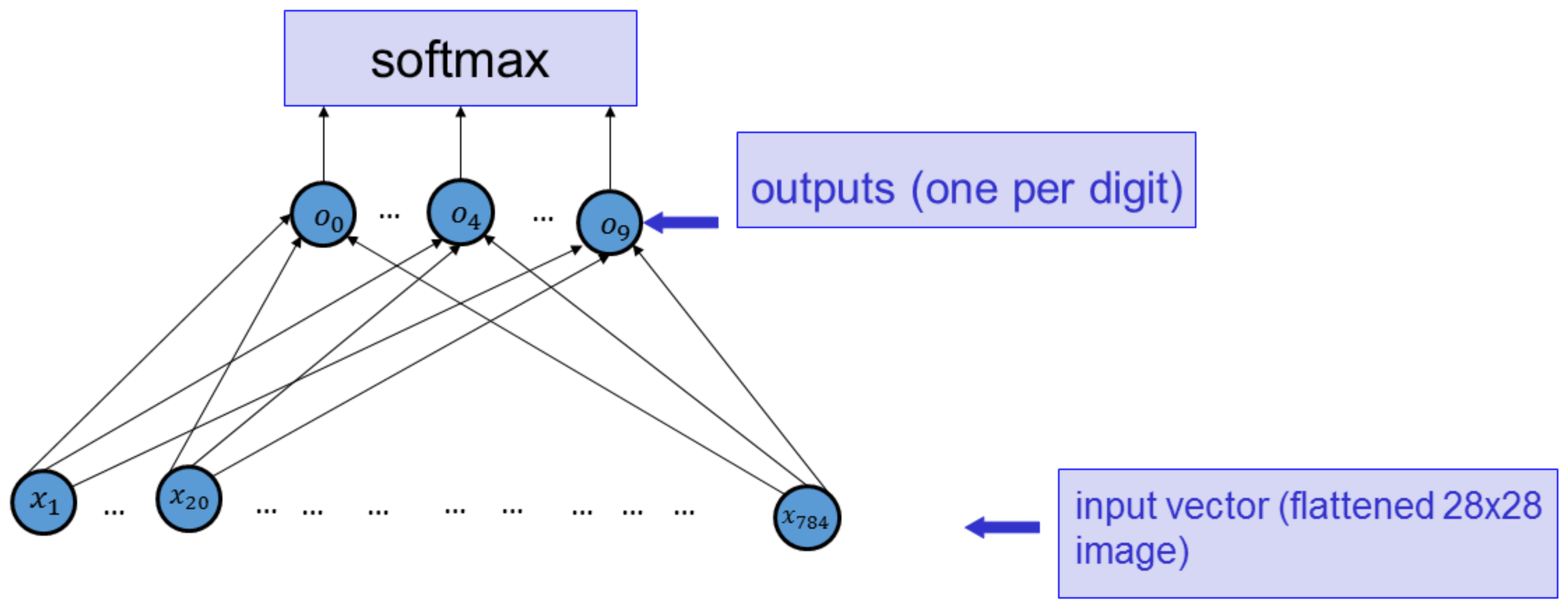
There's some code here (data: snapshot50.pkl). The code is not meant to be run as is, just to provide snippets. For beginners I recommend instead consulting online resources for code with more explanations/discussion (ask me for suggestions if unsure). Ultimately you must write your own implementation but you can use snippets if you cite the source.

Part 1 (5%)

In your report, include 10 images of each of the digits. You may find matplotlib's subplot useful.

Part 2 (10%)

Implement a function that computes the network below.



The o's here should simply be linear combinations of the x's (that is, the activation function is the identity). Specifically, use o_i = SUM_j w_ji x_j + b_i. Include the listing of your implementation (i.e., the source code for the function; options for how to do that in LaTeX are here) in your report for this Part.

Part 3 (10%)

(we'll discuss mini-batch gradient descent in class in Week#10, and I'll post a PDF)

For this project, we will be using the the sum of the negative log-probabilities of the correct answer for the N training cases under consideration as the cost function. (I.e., the negative log-likelihood of the training set.)

Implement a function that computes the gradient of this cost function with respect to the parameters of the network (W and b), for a given subset of training cases. Include a listing of your implementation in your report for this Part. You can, but do not have to, vectorize your code.

Part 4 (5%)

Verify that you are computing the gradient in Part 3 correctly by computing it both using your function and using a finite-difference approximation for several coordinates of the W and b. In your report for this Part, include the listing of the code that you used and the output of the code (which should include both your precise gradient and the approximation).

You will not get credit for Part 3 if you do not attempt Part 4

Part 5 (15%)

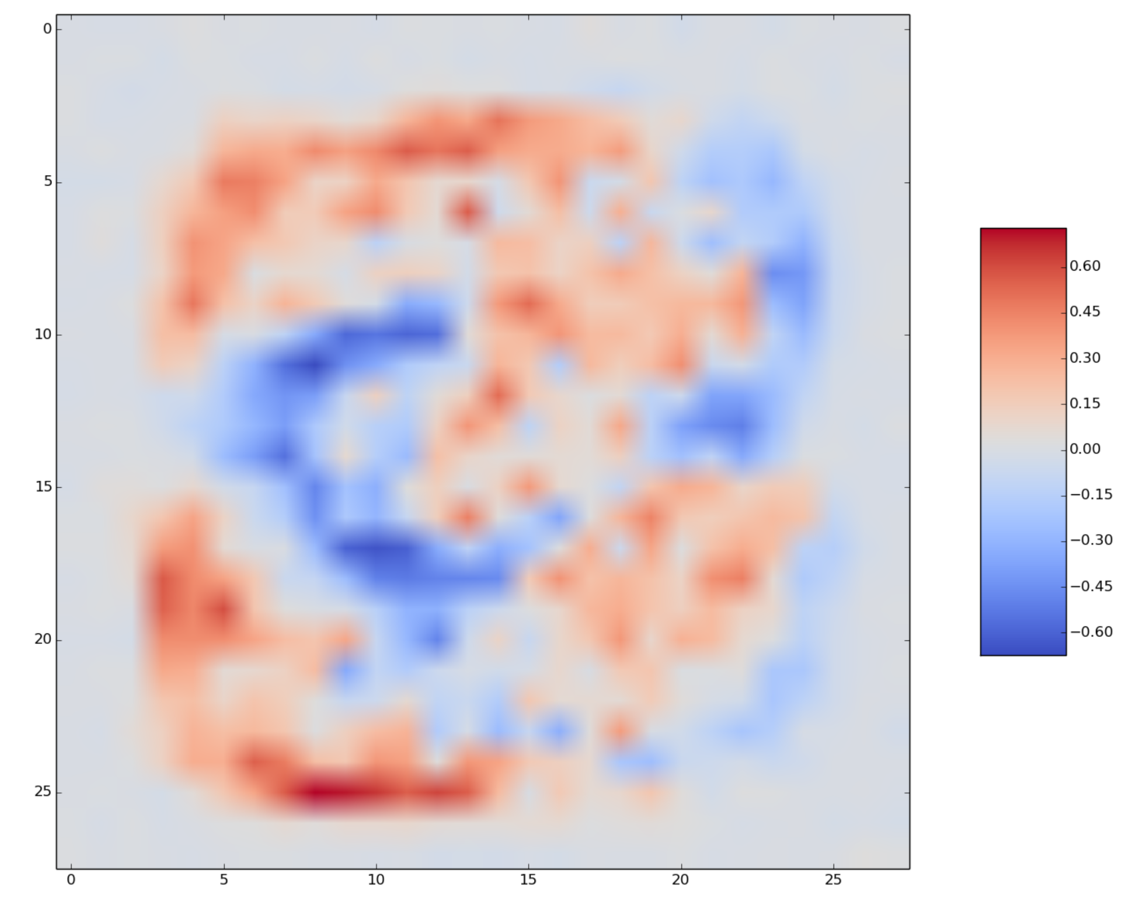
Write code to minimize your the cost function using mini-batch gradient descent, using the training set provided to you. You should be able to obtain test classification performance of over 91% correct classification (anything above 89% is OK). A learning rate of about 0.01 and batch sizes of 50 should work well.

For the training and the test set, graph the negative-log probability of the correct answer and correct classification rate versus the number of updates to the weights and biases during training. (I.e., plot the learning curves.)

In your report, display 20 digits which were classified correctly, and 10 digits from the test set which were classified incorrectly.

Part 6 (10%)

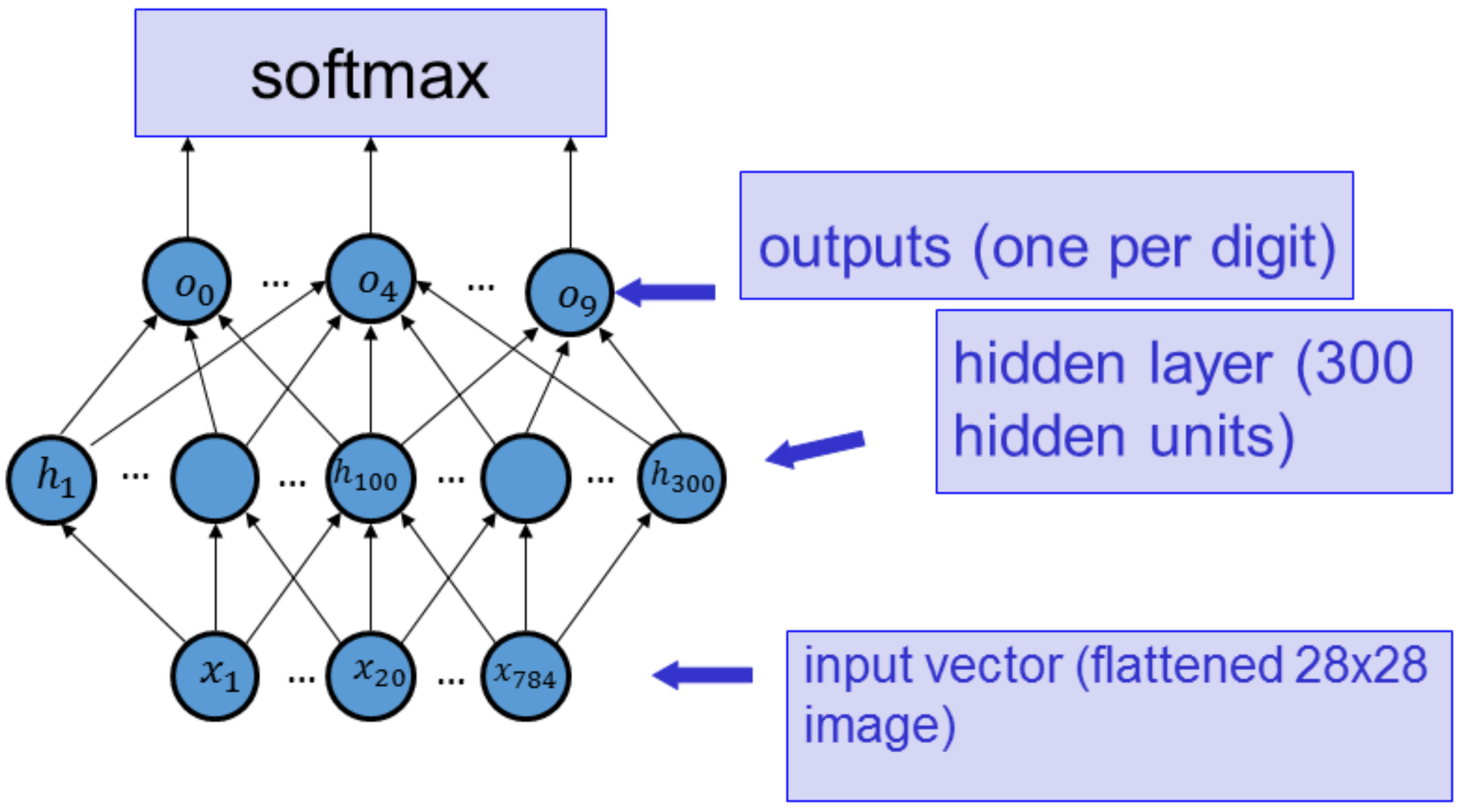
You can visualize what the network is doing by visualizing the W's as if they were digits. Visualize each of the set of W's that connect to o0, the set of W's that connect to o1, etc, with each w_ji displayed at an appropriate pixel. For example, for the W's that connect to o3, I obtain the following image.



Comment on the visualization of the weights that you obtained. (One or two sentences are enough: just tell us what you see.)

Part 7 (25%) but here you can use a high-level library such as Keras (or other choice, subject to approval)

For this part, you will implement a neural network with a hidden layer for digit classification. Specifically, you should implement the network sketched below, using tanh activation functions and 300 hidden units.



Part 8 (10%)

Use mini-batch gradient descent, using the training set provided to you. You should be able to obtain test classification performance of over 95% correct classification. A learning rate of about 0.01 and batch sizes of 50 should work well.

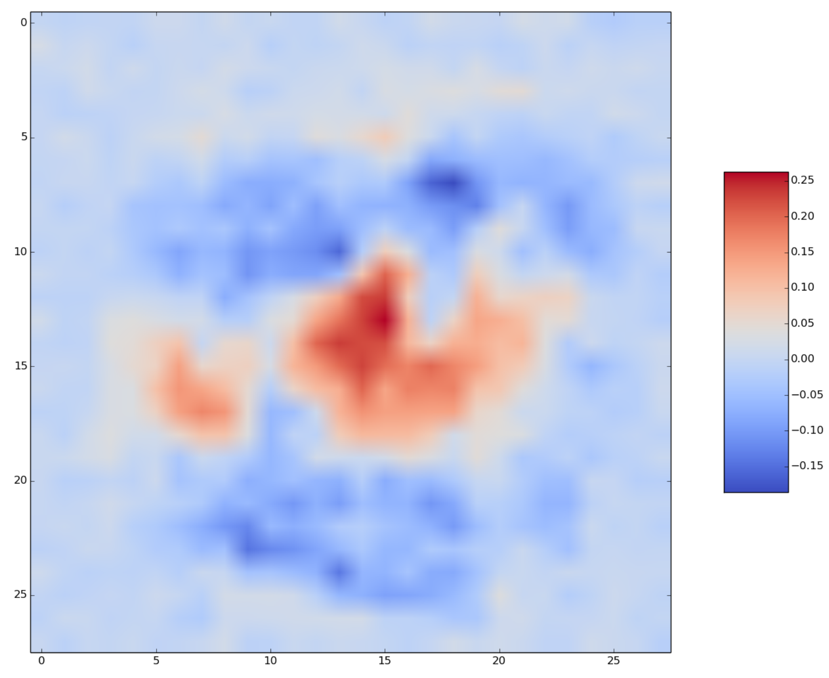
For the training and the test set, graph the correct classification rate versus the number of updates to the weights and biases during training. (I.e., plot the learning curves.)

In your report, display 20 digits which were classified correctly, and 10 digits from the test set which were classified incorrectly.

Part 9 (10%)

You can visualize what the network is doing by visualizing the W's *connected to the hidden layer* as if they were digits. Select two interesting W's to visualize (out of the total of 300) and explain what you think they are accomplishing. (The explanations should be different for the different W's).

For example, for one of the W connecting the inputs to the hidden layer, I obtain the following:



The weights connecting the hidden unit that corresonds to this W to the output units are: [-0.17553222, 0.09433381, -0.75548565, 0.13704424, 0.17520368, -0.02166308, 0.15751396, -0.31243968, 0.12079471, 0.66215879].

Your report should be in PDF format. You should use LaTeX to generate the report, and submit the .tex file as well. A sample template is on the course website. You will submit three files: digits.py, digits.tex, and digits.pdf. You can submit other Python files as well: we should have all the code that's needed to run your experiments.

or corresponding files in chosen language

Reproducibility counts! We should be able to obtain all the graphs and figures in your report by running your code. Submissions that are not reproducible will not receive full marks. If your graphs/reported numbers cannot be reproduced by running the code, you may be docked up to 20%. (Of course, if the code is simply incomplete, you may lose even more.) Suggestion: if you are using randomness anywhere, use numpy.random.seed(). We are fine with **minor** variations that arise due to the differences between your system and ours. For visualizing weights, it would be fine if we obtained different images that what you obtained when we run your code, but we should obtain *some* images.

or similar pseudorandom number generator

You must use LaTeX to generate the report. LaTeX is the tool used to generate virtually all technical reports and research papers in machine learning, and students report that after they get used to writing reports in LaTeX, they start using LaTeX for all their course reports. In addition, using LaTeX facilitates the production of reproducible results.

Important:

- Readability counts! If your code isn't readable or your report doesn't make sense, they are not that useful. In addition, we can't read them. You will lose marks for those things.
- It is perfectly fine to discuss general ideas with other people, *if you acknowledge ideas in your report that are not your own*. However, you must not look at other people's code, or show your code to other people, and you must not look at other people's reports, or show your report to other people. All of those things are academic offences.

in this paragraph "other people"
means outside your team