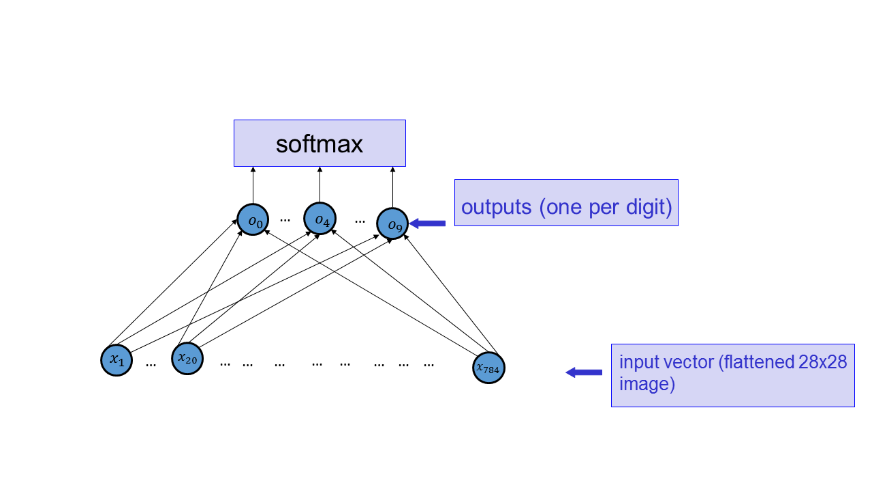
**Part 1 (5%)**

In your report, include 10 images of each of the digits.

**Part 2 (5%)**

Code: Implement a function that computes the network below.

For layer **o**: o\_i = SUM\_j w\_ji x\_j + b\_i.

Activation Function of o’s is the identity.

i.e. The set of o’s is the linear combinations of the x's

Report: the source code for the function

**Part 3 (10%)**

Cost Function: Cross-Entropy -

* sum of the negative log-probabilities of the correct answer for the N training cases under consideration as the cost function.

Code: 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.

Report: source code for function

**Part 4 (5%)**

Code: Implement function that verifies gradient correctness by computing it both using your function and using a finite-difference approximation for several coordinates of the W and b.

Report: source code for function & output of code (precise gradient and the approximation)

**Part 5 (15%)**

Code: Implement function to minimize the cost function using mini-batch gradient descent, using the training set provided to you. (Classification Performance > 91%) **learning rate** = 0.01

**batch sizes** = 50

Report:

For training & test set:

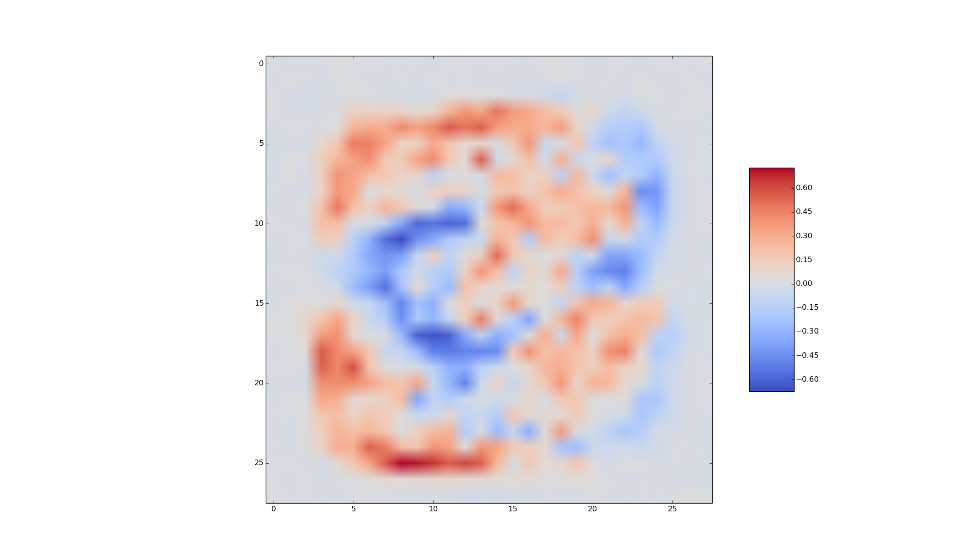
* Graph the number of updates (of W and b) vs negative-log probability (cost)
* Graph the number of updates (of W and b) vs correct classification rate (learning curve)
* display 20 digits from the test set which were classified correctly
* display10 digits from the test set which were classified incorrectly

**Part 6 (10%)**

Code: Visualize the W's as if they were digits.

For each digit:

* Visualize each of the set of W's that connect to o\_j

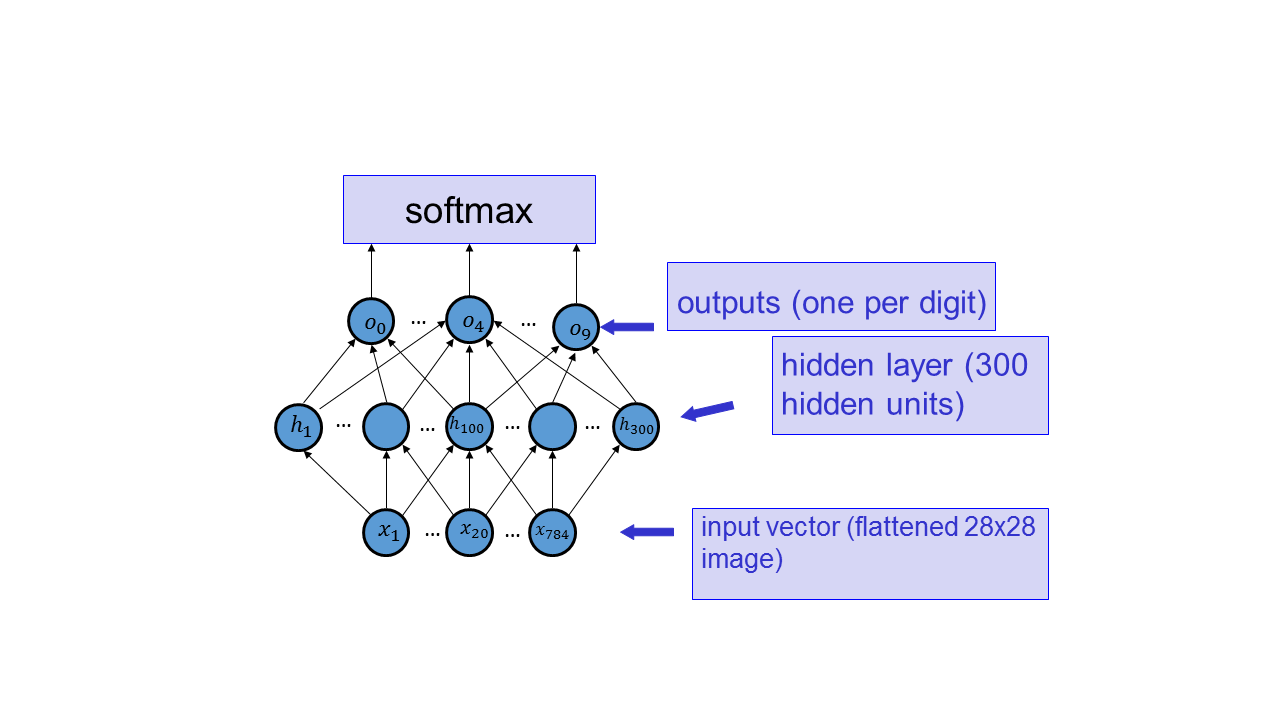


Comment on the visualization of the weights that you obtained. (One / two sentences)

**Part 7 (25%)**

Code: Implement a neural network with a hidden layer for digit classification.

Use [tanh](http://math2.org/math/derivatives/more/hyperbolics.htm) activation functions and 300 hidden units.



Implement a function that computes the gradient of the negative log probability of the correct answer function, computed for a set of training examples \*\* vectorized

Report: source code & explanation for why each line of the code actually works

i.e., provide the computation using sigma notation, and briefly show how the matrix algebra accomplishes the same thing. For this part, you may include scanned neatly handwritten explanation included in the pdf.

**Part 8 (5%)**

Code: Implement function that verifies gradient correctness by computing it both using your function and using a finite-difference approximation for several coordinates of the W and b.

Report: source code & output (precise gradient & approximation)

**Part 9 (10%)**

Code: Implement function to minimize the cost function using mini-batch gradient descent, using the training set provided to you. (Classification Performance > 95%) **learning rate** = 0.01

**batch sizes** = 50

Report:

For training & test set:

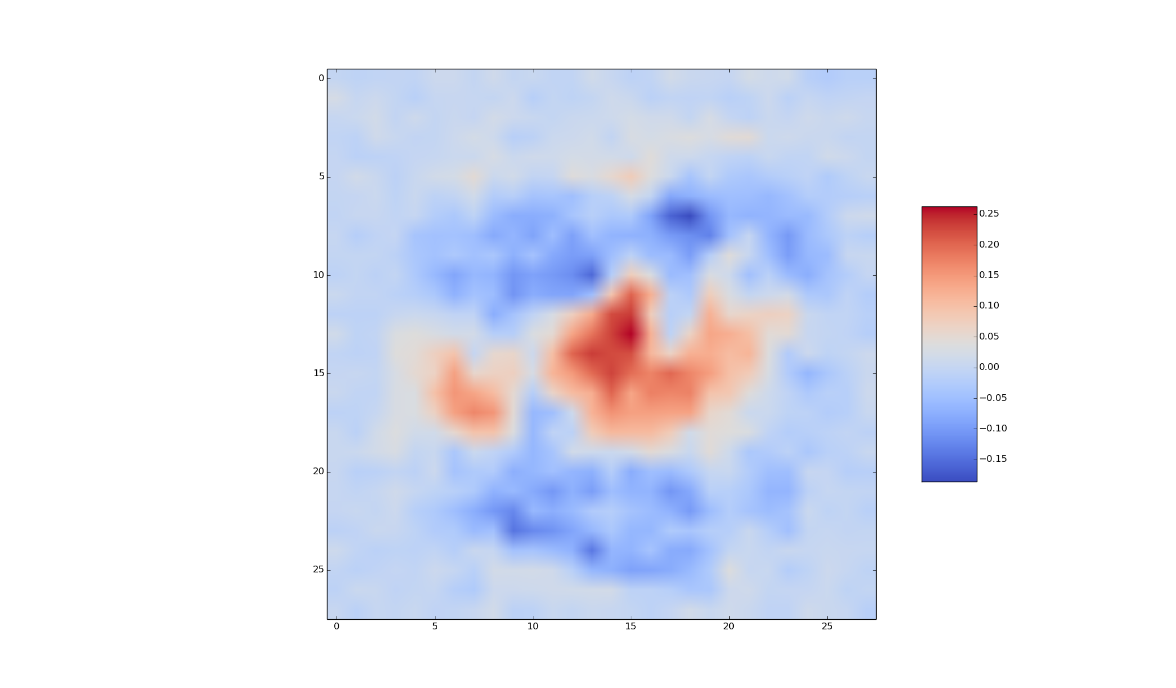
* Graph the number of updates (of W and b) vs negative-log probability (cost)
* Graph the number of updates (of W and b) vs correct classification rate (learning curve)
* display 20 digits from the test set which were classified correctly
* display10 digits from the test set which were classified incorrectly

**Part 10 (10%)**

Report: Visualize 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 corresponds 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].