Homework4 / Handwritten Digits Recognition

李豪韋 (HW-Lee) ID 103061527

Overview

Neural nets can theoretically learn any function, so we can also utilize them to do classification tasks. In recent years, automatic recognition systems have been getting more important and more widely used. Therefore, we try to implement a system that is able to recognize hand-written digits. Obviously, it is a classification problem, all things we have to do are 1) separating instances which belong to different classes as much as possible, 2) aggregating instances which belong to the same class as close as possible, 3) suppressing the confidence of outputs corresponding to wrong classes, and 4) lifting the confidence of the output corresponding to the right class. In this project, the task has been implemented and the average accuracy can achieve close to 90%. The more detailed information of the code is publicly available on https : //github.com/HW-Lee/2015-NN-Homeworks/tree/master/HW04.

Implementation

1. **Net Structure**

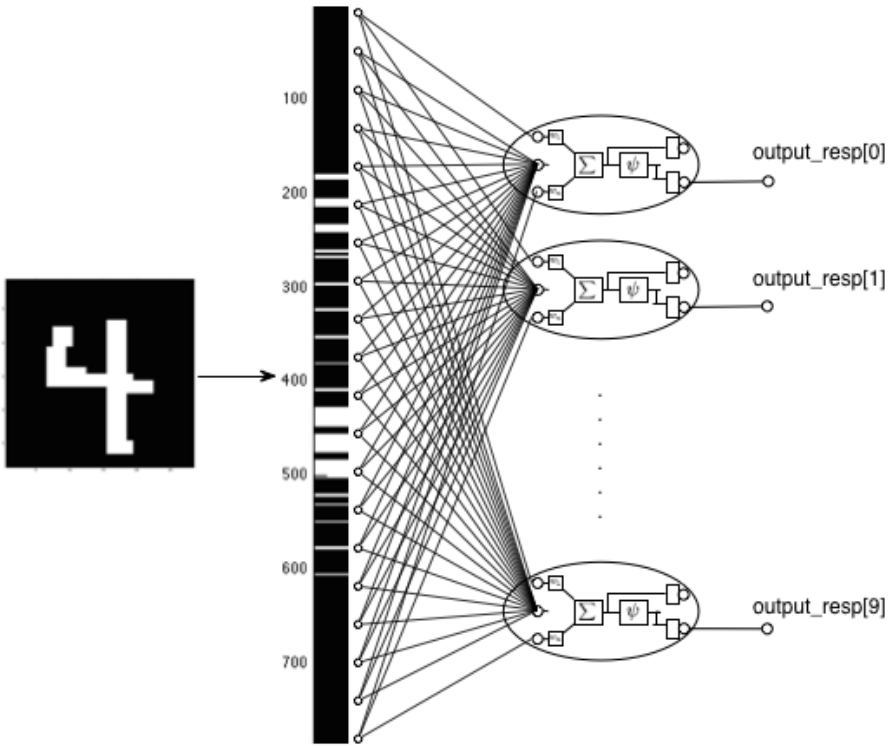

Figure 1: The structure of net. It simply consists of 10 neurons without any hidden unit, which acts like a linear filter, and the image data will be reshaped from 2D to 1D then all pixels will be regarded as an input port. However, the only thing different from the linear filter implemented in Hw2 is that the linear outputs are cascaded to the logistic function that maps any real value into a value ranged from 0 to 1, in order to make the output a measure of 'confidence'. Finally, the prediction is followed by the rule of choosing the class which turns out the highest confidence.

2. **Process**

   - **Weights Initialization**: make an initial guess.
   - **Randomly Feeding Instances**: randomly choose an instance with its ground truth.
   - **Stochastic Gradient Descent**: Use SGD to obtain a good set of weights.

Results

1. **Confusion matrix with cross-validation (average accuracy is 0.877)**

truth \ result	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
'0'	**0.99**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
'1'	0.00	**0.94**	0.02	0.00	0.00	0.02	0.00	0.00	0.02	0.00
'2'	0.01	0.01	**0.87**	0.00	0.01	0.00	0.01	0.03	0.05	0.01
'3'	0.01	0.00	0.04	**0.84**	0.01	0.04	0.02	0.03	0.01	0.00
'4'	0.00	0.01	0.03	0.01	**0.85**	0.01	0.02	0.03	0.02	0.02
'5'	0.05	0.00	0.00	0.07	0.03	**0.79**	0.01	0.00	0.04	0.01
'6'	0.00	0.00	0.00	0.01	0.01	0.03	**0.94**	0.00	0.01	0.00
'7'	0.01	0.01	0.01	0.00	0.01	0.00	0.00	**0.93**	0.00	0.03
'8'	0.01	0.03	0.01	0.04	0.01	0.00	0.01	0.02	**0.85**	0.02
'9'	0.04	0.02	0.01	0.02	0.06	0.02	0.01	0.04	0.01	**0.77**

2. **Confusion matrix with empirical data (average accuracy is 0.9124)**

truth \ result	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
'0'	**0.98**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
'1'	0.00	**0.96**	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00
'2'	0.01	0.01	**0.87**	0.01	0.02	0.00	0.02	0.02	0.02	0.01
'3'	0.00	0.01	0.03	**0.87**	0.00	0.03	0.01	0.01	0.01	0.02
'4'	0.00	0.01	0.00	0.00	**0.92**	0.00	0.01	0.00	0.00	0.04
'5'	0.02	0.01	0.01	0.03	0.01	**0.86**	0.02	0.00	0.03	0.01
'6'	0.01	0.00	0.00	0.00	0.00	0.01	**0.97**	0.00	0.00	0.00
'7'	0.01	0.01	0.01	0.00	0.02	0.00	0.00	**0.93**	0.00	0.02
'8'	0.01	0.01	0.02	0.02	0.01	0.02	0.02	0.01	**0.88**	0.01
'9'	0.02	0.00	0.01	0.02	0.02	0.01	0.00	0.04	0.00	**0.89**

3. **Average response with different classes**

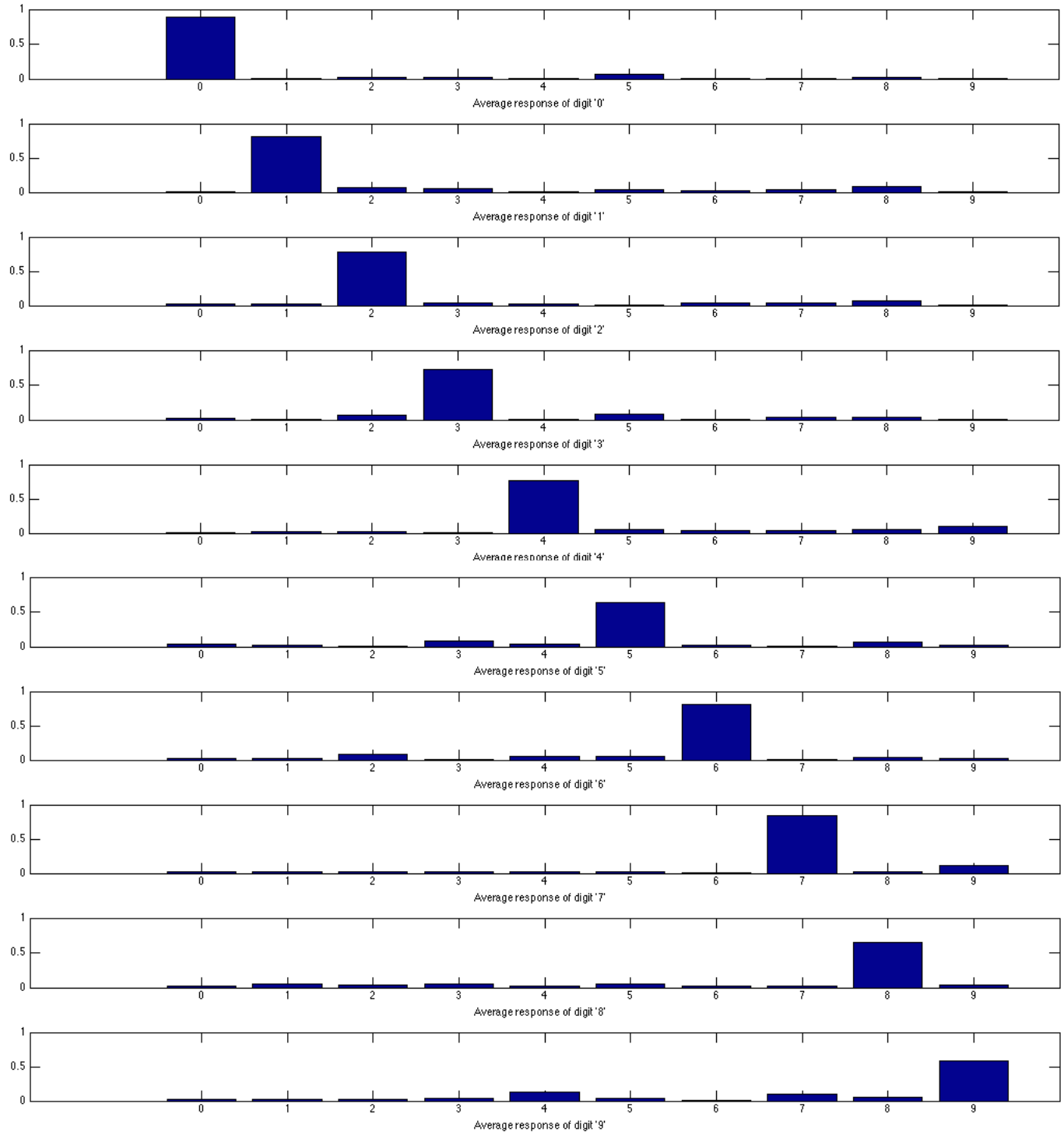

Figure 2: the response evaluated with 100 randomly sampled images of each class

Discussion

1. What is a good initialization strategy?

   Actually, any initialization method will not affect the performance too much because the structure of my system is not complex, most of issues concerned in back-propagation applied to multi-layer net are not critical in gradient descent of linear filters (e.g. no zero-initialization, random variation). However, an appropriate initial guess will decrease lots of computation time for learning with training sets. In practice, a system with better initial weights gets converged after 30,000 iterations while that with randomly initialized weights gets converged after more than 100,000 iterations. Then, what is a good set of initial weights? I use the training strategy described in the homework description sheet, namely simply averaging all training data of each class, I use it as the initial weights. It is a good initial guess such that the system can get converged only with 30,000 iterations!

2. How to define input space?

   As the sheet described, every image only contains raw data with value either 0 or 1. If we only apply it as the input space directly, the offset(0) can not be used as a 'penalty' while training. (because the value 0 is constant with any arbitrary weight) For enabling the sensitivity of offset, the data are subtracted by 0.5 for the purpose of making data ranged in ± 0.5. Hence, the offset is as sensitive as the onset and the input space is balanced. (0-centred range)

3. Something interesting of final weights.

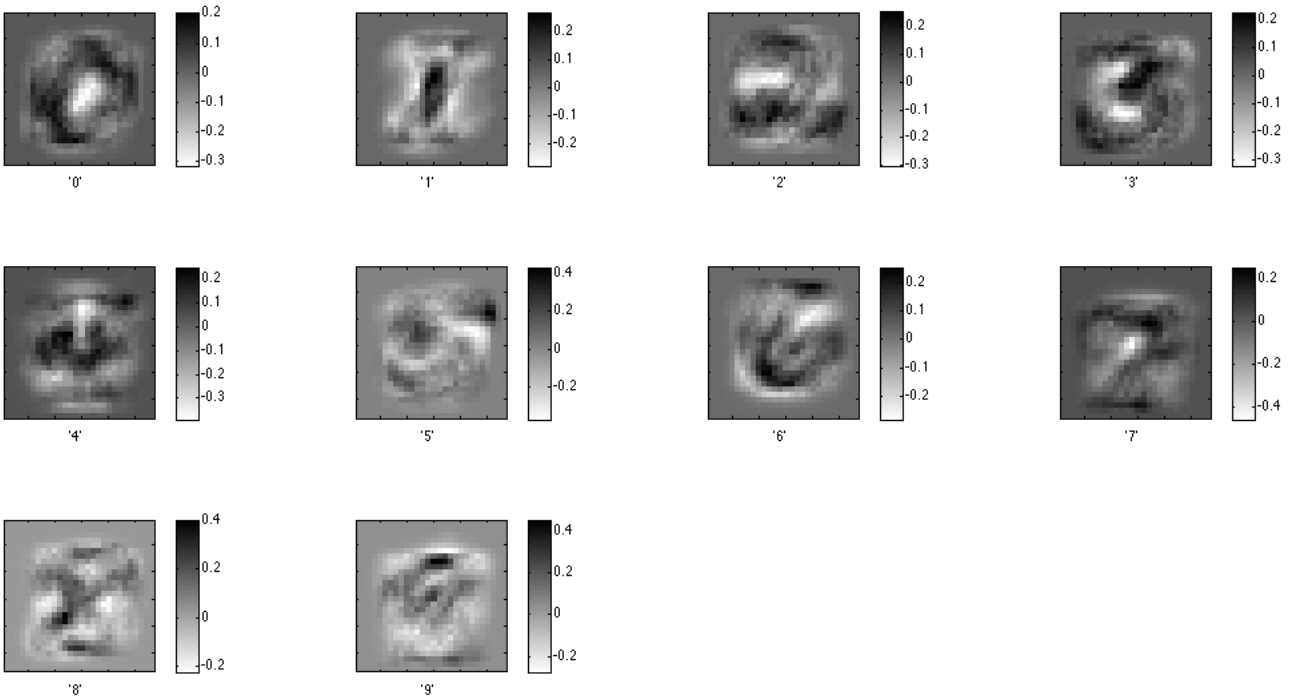

Figure 3: Final weights reshaped to 2D (with the same structure as that of images) of each class. There comes some intuitive and interesting conclusions: 1) The final weights get converged to the results which are 'visually similar' to the appearance of digits we think. 2) The brighter parts, which refers to relatively smaller values, in each class are also features that people use to recognize a digit. (e.g. we expect that there is a 'hole' at the center of '0', and there are two vertically arranged holes in '8'.)