Exercise: Self Taught Learning

Overview

In this exercise, we will use the self-taught learning paradigm with convolutional nerual network, RICA and softmax classifier to build a classifier for handwritten digits.

You will be building upon your code from the earlier exercises. First, you will train your RICA on patches extracted from an "unlabeled" training dataset of handwritten digits. This produces filters that are penstroke-like. We then extract features from a labeled dataset of handwritten digits by convolving with these learnt filters. These features will then be used as inputs to the softmax classifier that you wrote in the previous exercise.

Concretely, for each example in the the labeled training dataset x_l, we forward propagate the example through a convolutional and a pooling layer to obtain the activation of the hidden units $a^{(2)}$. We now represent this example using $a^{(2)}$ (the "replacement" representation), and use this to as the new feature representation with which to train the softmax classifier.

Finally, we also extract the same features from the test data to obtain predictions.

In this exercise, our goal is to distinguish between the digits from 0 to 4. We will use an "unlabeled" dataset with all 10 digits to learn the filters; we will then use a labeled dataset with the digits 0 to 4 with which to train the softmax classifier.

In the starter code, we have provided a file "`stlExercise.m`" that will help walk you through the steps in this exercise.

Dependencies

The following additional files are required for this exercise:

- MNIST Dataset (http://yann.lecun.com/exdb/mnist/)
- The **stl** (https://github.com/amaas/stanford_dl_ex/tree/master/stl) folder of the exercises starter code

You will also need your code from the following exercises:

- RICA (http://ufldl.stanford.edu/tutorial/unsupervised/ExerciseRICA)
- Softmax Regression (http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression)
- Convolutional Neural Network (http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionalNeuralNetwork)

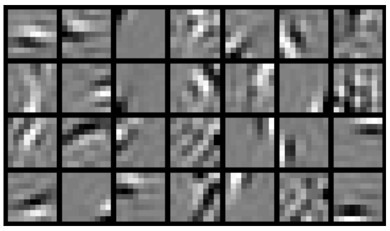
If you have not completed the exercises listed above, we strongly suggest you complete them first.

Step 1: Generate the input and test data sets

Go to the `stl` folder of the exercises code, which contains starter code for this exercise. Additionally, you will need to download the datasets from the MNIST Handwritten Digit Database for this project.

Step 2: Train RICA

In the starter code, we have provided code to split the MNIST dataset into 50000 "unlabelled" images and 10000 "labelled" images. We also provide code to randomly extract 200000 8-by-8 patches from the unlabelled dataset. You will need to whiten the patches using the `zca2.m` function seen in the RICA exercise. Then you will train an RICA on the 200000 patches, using the same `softICACost.m` function as you had written in the previous exercise. (From the earlier exercise, you should have a working and vectorized implementation of the RICA.) For us, the training step took less than 25 minutes on a fast desktop. When training is complete, you should get a visualization of pen strokes like the image shown below:



Informally, the features learned by the RICA should correspond to edge detectors.

Step 3: Extracting features

After the RICA is trained, you will use it to extract features from the labelled handwritten digit images. To extract features from an image of hand-written digit, you will first convolve the learnt RICA weights with the image, followed by RICA-style square-square-root pooling on the response.

Complete `feedForwardRICA.m` to produce a matrix whose columns correspond to activations of the hidden layer for each example, i.e., the vector $a^{(2)}$ corresponding to activation of layer 2. (Recall that we treat the inputs as layer 1).

Step 4: Training and testing the softmax regression model

Use your code from the softmax exercise (`softmax_regression_vec.m`) to train a softmax classifier using the training set features (`trainFeatures`) and labels (`trainLabels`).

Step 5: Classifying on the test set

Finally, complete the code to make predictions on the test set (`testFeatures`) and see how your learned features perform! If you've done all the steps correctly, you should get **100%** train accuracy and ~**99%** test accuracy. As a comparison, we get **97.5%** test accuracy with random convolutional weights. Actual results may vary as a result of random initializations