Programming assignment #3:
Multilayer Neural Network

4/17/2015

In a group of 2-3
Due: May 11th, Monday

This assignment derives from Stanford UFLDL tutorial (Multilayer Neural Network, Supervised Neural Networks). You can check the tutorial for more information about the backprop algorithm.

Exercise

For this assignment, you will implement a supervised multilayer neural network architecture.

Use the github repository as a starter code. (**Codes in the repository have changed since last week, so you have to re-download it and not use the starter codes from last assignment's folder. Sorry about that!**) Only codes in folder *multilayer_supervised/* is useful in this exercise. The code is in matlab, so you are encouraged to use matlab for this assignment. Using other languages is okay, but if you use another language, you will also need to use l-bfgs method in that language (see below). There are l-bfgs packages available online in different languages.

Dataset

The datasets we are using for this assignment this time are POFA dataset and NimStim dataset. They are both face datasets.You are going to use the neural network to do two tasks: identity recognition, facial expression recognition. You'll need to download the datasets from the links. **Note that these are proprietary datasets. Please do not make them available in any way except to your team members, and destroy them after use.**

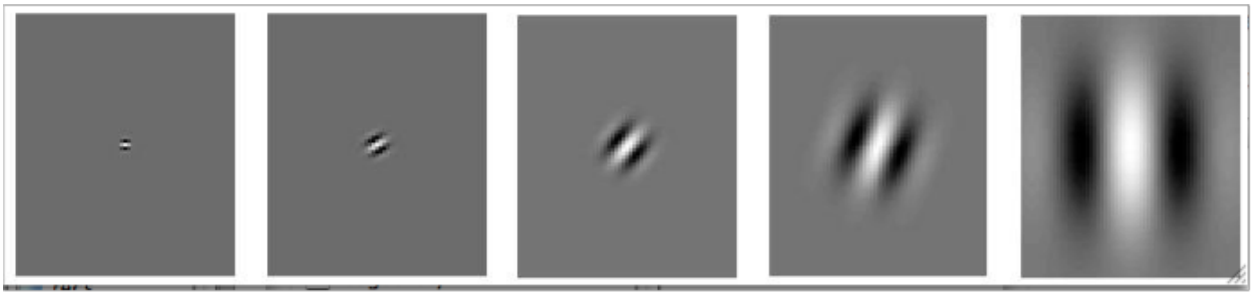
For the POFA dataset, the name of files is in this form: ???-??-?-*.pgm (eg. aa1-AN-F-14). First two letters indicate identity, second part consisting of two letters indicates facial expression (AN-anger, DI-disgust, HA-happiness, SP-surprise, SA-sadness, FE-fear), and third part with one letter is gender (F-female, M-male). This dataset is labeled by me and my colleague, so if you find something weird, please tell me.

It's similar for NimStim. ???-??-?.bmp (eg. 23M_An_C.bmp) First two digits indicate identity. Second part consisting of two letters indicate facial expression but you don't need that in this exercise.

Identity recognition

For this problem, you will use the NimStim dataset. You need to do:

1. Preprocessing of the data. The preprocessing steps derive from EMPATH, please read this to get the idea of this step.
    a. **Train/test separation.** Separate train and test images. Make sure images are randomly permuted, and make the training:test ratio=3:1.
    b. **Gabor filtering.** For each image:
        i. load image and change it from RGB to grayscale
        ii. Resize image to a certain size (e.g. 64*64)
        iii. Use gabor filters to filter the image. Your gabor filters should resemble these (these are the cosine parts of the filters):



You can refer to gabor filter codes online. But be careful the filter looks like above picture. (For example, to be biologically plausible, the filters should look a certain way - with just a few lobes)
The gabor filter can be formed like:

$$g(x, y, \theta, k, \sigma) = exp\{j \cdot k[cos(\theta)x + sin(\theta)y]\} \cdot exp(-\frac{k^2(x^2+y^2)}{2\sigma^2})$$

where (x,y) *is the pixel location,* θ *represents orientation,* k *is scale, and* σ *is the standard deviation of Gaussian envelope. You need to do gabor filtering on 5 different scales and 8 orientations (equally spaced from 0 to* $7\pi/8$*). That means, you have 40 different filters. Here is a recommended set of parameters:* $k(i) = \frac{2\pi}{64} \times 2^i,\ i = 1, 2, .., 5$*;* $\sigma = \pi$*. Size of gabor filter: 96. Use the magnitude of the Gabor filters as the final result, i.e., square the sine and cosine responses, sum them, and take the square root.*

        iv. After filtering, you get 40 Gabor images, one for each filter. **Downsample** the images to a smaller size (e.g. 8*8), and take absolute value of the complex filtered images.
        v. **z-score** the filtered images on an individual basis. I.e., one scale and one orientation at one location gets z-scored.

c. **PCA.** On the dataset, for each scale, do:
    i. stretch the 8 filtered images into a vector.
    ii. Do PCA on the vectors and reduce the dimensionality of feature vector. Recommended dimension remaining from each scale: 8, giving 40 principal components in total. [You can use turk and pentland trick to speed up the PCA calculation. See here in p75.]
d. **z-score.** Now, for each image, we have 5 feature vectors (of 5 scales). Concatenate them and z-score them on every dimension. Note that they are already 0-mean. So all you need to do is divide by the square root of the eigenvalues.

2. Use the *multilayer_supervised/run_train.m* as a reference. Build your neural network and estimate the parameters $\{w, b\}$. Do:
    a. minFunc uses L-BFGS for optimization given the objective function and gradient in *multilayer_supervised/supervised_dnn_cost.m*. It's your job to create a cost function which does forward propagation of your neural network, and computes gradients in *multilayer_supervised/supervised_dnn_cost.m*. Requirement:
        i. Your implementation should support training neural networks with multiple hidden layers with arbitrary numbers of hidden units.
        ii. For nonlinear activations, you are free to choose from logistic and tanh.
        iii. Use cross-entropy loss, the same as with softmax regression.
        iv. In your implementation, you should include momentum.
        v. Include l2-regularization.
    b. Numerically check the gradients and make sure your implementation computes the gradient correctly.
    c. Build a network with one hidden layer. Use preprocessed data from 1) as input. Decide the number of hidden units and other hyperparameters (e.g. regularization). Initialization of weights and bias has been done in *multilayer_supervised/initialize_weights.m* (You're welcome to try different initializations). Train and test the network. Draw the plot of convergence (objective function value - iteration).
    d. Implement your stochastic gradient descent method. Do c) for SGD method.

    In your implementation, take advantage of matlab's quick matrix operation. Full credit will be given with vectorization.

3. Finally, evaluate the two methods by training time and accuracy of test data.

Facial expression recognition

This exercise is similar to identity recognition, but here we use POFA dataset. There are 14 actors in the dataset. This time train the expression recognition system using leave-one-out cross-validation.

For SGD, do with a hold-out validation set for early stopping: A given network is trained on all the images of 12 of the 14 actors in POFA, and use 13th as hold-out set to determine when to stop (Training is stopped when the error on the hold-out set is minimized.). After training is stopped, the 14th actor is used for testing. This is repeated 14 times for each actor as test set. You can randomly choose a hold-out validation set for each round. Refer to the EMPATH paper for more details. (But note they did in a more complicated way: they performed training runs with every possible combination of generalization and hold out sets, for a total of 182 (14 by 13) individual networks. We don't have to do the 13 different choices of validations for each test)

For l-bfgs, use 1 actor as test set and other 13 as training set. Repeat this 14 times.

This exercise derives from EMPATH, please read this to get the idea of the methods and how cross validation is done.

You need to do:

1. Preprocessing of the data.
    a. **Train/test separation.** Separate train and test images as mentioned above.
    b. **Gabor filtering.** same as before.
    c. **PCA.** same as before. Note you need to choose a proper dimension size this time.
    d. **z-score.** same as before.

2. Again, use the *multilayer_supervised/run_train.m* as a reference.
    a. Design your network. Use both MinFunc and SGD for training.
    b. Draw the plot of convergence. (This step you don't have to do 14 times, only once is enough).
3. Finally, evaluate the two methods by training time and accuracy of test data.

Report

Your report should be in NIPS format. (Style in latex)

In your report, include:

1. Description of approach. Make sure you include:

   a. Plot the gabor filters in 8 orientations, 5 scales. For preprocessing, if you use different set of parameters with suggested values, make it clear.
   b. Description of forward & backward propagation steps to calculate objective function and gradient in both sum-of-nodes form and vectorized form. Remember to include l2 regularization here.
   c. Description of your stochastic gradient descent, you can write pseudo codes for this (Specify important parameters, such as momentum, stopping criterion, learning rule, tricks you use).
2. Results. For each optimization method, provide:
   a. plot for speed of convergence during training procedure (objective function value - # of iteration).
   b. training time.
   c. accuracy on training and test set.
3. Discussion of your results, including any obstacles overcome or remaining bugs.
4. At the end, there should be a paragraph by each team member outlining their contribution and what they learned on the project.

Submission

1. **Each group submit one report and all your codes to yuw176@ucsd.edu**
2. **Make your title [291Programming_Assignment#3]**