CSC 411/2515
Introduction to Machine Learning
Assignment 2
Out: Oct. 26
Due: Nov. 9 [10 AM]

Overview

In this assignment, you will first derive the learning rule for mixture of Gaussians models and convolutional neural networks (CNN), and then experiment with these models on a subset of the Toronto Faces Dataset (TFD) [1]. Some code that partially implements a regular neural network, a convolutional neural network, and a mixture of Gaussians model is available on the course website (in python).

We subsample 3374, 419 and 385 grayscale images from TFD as the training, validation and testing set respectively. Each image is of size 48×48 and contains a face that has been extracted from a variety of sources. The faces have been rotated, scaled and aligned to make the task easier. The faces have been labeled by experts and research assistants based on their expression. These expressions fall into one of seven categories: 1-Anger, 2-Disgust, 3-Fear, 4-Happy, 5-Sad, 6-Surprise, 7-Neutral. We show one example face per class in Figure 1.

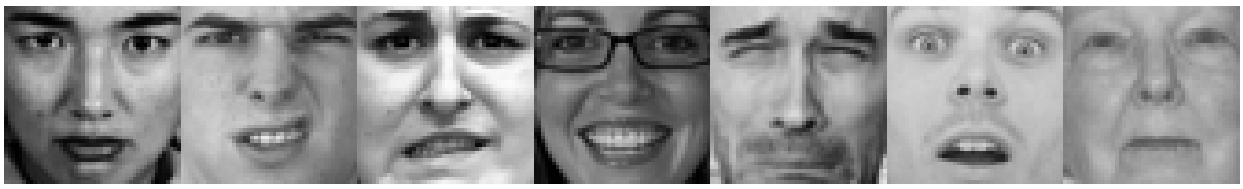

Figure 1: Example faces. From left to right, the the corresponding class is from 1 to 7.

1 EM for Mixture of Gaussians (10 pts)

We begin with a Gaussian mixture model:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1}$$

Consider a special case of a Gaussian mixture model in which the covariance matrix $\boldsymbol{\Sigma}_k$ of each component is constrained to have a common value $\boldsymbol{\Sigma}$. In other words $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$, for all k. Derive the EM equations for maximizing the likelihood function under such a model.

[1] `http://aclab.ca/users/josh/TFD.html`

2 Convolutional Neural Networks (10 pts)

Let $x \in \mathbb{R}^{N \times H \times W \times C}$ be N images, and $f \in \mathbb{R}^{I \times J \times C \times K}$ be the convolutional filters. H, W are the height and width of the image; I, J are the height and width of the filters; C is the depth of the image (a.k.a. channels); K is the number of filters.

Padding is an operation that adds zeros to the edges of an image to form a larger image. Formally, the padding operator `pad` is defined as:

$$x^{(P,Q)} = \texttt{pad}(x, P, Q) \in \mathbb{R}^{N \times (H+P) \times (W+Q) \times C} \tag{2}$$

$$x^{(P,Q)}_{n,h,w,c} = \begin{cases} x_{n,h-\lfloor \frac{P}{2} \rfloor, w-\lfloor \frac{Q}{2} \rfloor, c} & \text{if } \lfloor \frac{P}{2} \rfloor + 1 \leq h \leq \lfloor \frac{P}{2} \rfloor + H, \lceil \frac{Q}{2} \rfloor + 1 \leq w \leq \lfloor \frac{Q}{2} \rfloor + W \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Define the 2-D convolution operator $*$ as:

$$y_{n,h',w',k} = x * f = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{c=1}^{C} x_{n,h'+i,w'+j,c} \cdot f_{i,j,c,k} \tag{4}$$

for $1 \leq h' \leq H - I + 1$, $1 \leq w' \leq W - J + 1$. (Note: this is "correlation" in signal processing.)

Define the filter transpose as:

$$f^{\top}_{i,j,k,c} = f_{I-i+1,J-j+1,c,k} \tag{5}$$

Given the forward propagation equation $y = x^{(I-1,J-1)} * f$, and given some loss function E, show that the update equations for the filters and activities have the following form:

$$\left(\frac{\partial E}{\partial f}\right)_{c,i,j,k} = x^{(I-1,J-1)}_{c,h,w,n} * \left(\frac{\partial E}{\partial y}\right)_{h,w,n,k} \tag{6}$$

$$\left(\frac{\partial E}{\partial x}\right)_{n,h,w,c} = \left(\frac{\partial E}{\partial y}\right)^{(I-1,J-1)}_{n,h,w,k} * f^{\top} \tag{7}$$

For this exercise, you may assume $n = 1, c = 1, k = 1$, but you should use the equations above to implement the CNN later.

(Hint: It may be easier to convert $*$ to matrix multiplication. You may start with a simple example with $H = W = 5$, $I = J = 3$.)

3 Neural Networks (40 points)

Code for training a neural network (fully connected and convolutional) is partially provided in the following files.

- `nn.py` : Train a fully connected neural network with two hidden layers.
- `cnn.py` : Train a convolutional neural network (CNN) with two convolutional layers and one fully connected output layer.

You need to fill in some portion of the code for:

- Performing backward pass of the network.
- Performing weight update with momentum.

First, follow the instruction in the files to complete the code.

3.1 Basic generalization [5 points]

Train a regular NN and a CNN with the default set of hyperparameters. Examine the statistics and plots of training error and validation error (generalization). How does the network's performance differ on the training set versus the validation set during learning? Show a plot of error curves (training and validation) for both networks.

3.2 Optimization [10 points]

Try different values of the learning rate ϵ ("eps"). Try 5 different settings of ϵ from 0.001 to 1.0. What happens to the convergence properties of the algorithm (looking at both cross-entropy and percent-correct)? Try 3 values of momentum from 0.0 to 0.9. How does momentum affect convergence rate? Try 5 different mini-batch sizes, from 1 to 1000. How does mini-batch size affect convergence? How would you choose the best value of these parameters? In each of these hold the other parameters constant while you vary the one you are studying.

3.3 Model architecture [10 points]

Fix momentum to be 0.9. Try 3 different values of the number of hidden units for each layer of the fully connected network (range from 2 to 100), and 3 values of the number of filters for each layer of the conv-net (range from 2 to 50). You might need to adjust the learning rate and the number of epochs. Comment on the effect of this modification on the convergence properties, and the generalization of the network.

3.4 Compare CNNs and fully connected networks [10 points]

Calculate the number of parameters (including biases) in each type of network. Compare the performance of a conv-net and a regular network for the similar number of parameters. Which one leads to better generalization and why? Plot the first layer filters of the CNN, and also plot the first layer weights of the fully connected neural network. Briefly comment on the visualization.

3.5 Network Uncertainty [5 points]

Plot some examples where the neural network is not confident of the classification output (the top score is below some threshold), and comment on them. Will the classifier be correct if it outputs the top scoring class anyways?

4 Mixtures of Gaussians (40 points)

4.1 Code

The file `mogEM.py` implements methods related to training MoG models.
The file `kmeans.py` implements k-means.
As always, read and understand code before using it.

4.2 Training (10 points)

Train a mixture-of-Gaussians using the code in `mogEM.m`. Let the number of clusters in the Gaussian mixture be 7, and the minimum variance be 0.01. You will also need to experiment with the parameter settings, e.g. `randConst`, in that program to get sensible clustering results. And you'll need to execute `mogEM` a few times and investigate the local optima the EM algorithm finds. Choose a good model for visualize your results. To visualize, after training, show both the mean vector(s) and variance vector(s) as images and show the mixing proportions for the clusters. Finally, provide the training curve of log-likelihood. Look at `kmeans.py` to see an example of how to do this in Python.

4.3 Initializing a mixture of Gaussians with k-means (10 points)

Training a MoG model with many components tends to be slow. People have found that initializing the means of the mixture components by running a few iterations of k-means tends to speed up convergence. You will experiment with this method of initialization. You should do the following.

- Read and understand `kmeans.py`.
- Change the initialization of the means in `mogEm.py` to use the k-means algorithm. As a result of the change, the model should run k-means on the training data and use the returned means as the starting values for `mu`.
- Train a MoG model with 7 components on all training data using both the original initialization and the one based on k-means (set iteration number of k-means to be 5). Show the training curves of log-likelihood and comment on the speed of convergence resulting from the two initialization methods.

4.4 Classification using MoGs (20 points)

Now we will investigate using the mixture of Gaussian models for classification. The goal is to decide which expression class d a new input face image $\mathbf{x}$ belongs to. Specifically, we only work with two classes of expressions: 1-Anger, 4-Happy. For each class, we can train a mixture of Gaussian model on examples from that class. After training, the likelihoods $p(\mathbf{x}|d)$ can be computed for an image $\mathbf{x}$ by consulting the trained model; probabilistic inference can be used to compute $p(d|\mathbf{x})$, and the most probable expression class can be chosen to classify the image (Hint: using Bayes Theorem and note that $p(\mathbf{x})$ can be regarded as constant).

(a) (10 points) Write a program that computes the error rate of classification based on the trained models. You can use the method `mogLogLikelihood` in `mogEm.py` to compute the log-likelihood of examples under any model. You will compare models trained with the different number of mixture components, specifically, 7, 14, 21, 28 and 35. Plot the results. The plot should have 3 curves of classification error rates versus number of mixture components:

- The classification error rate, on the training set
- The classification error rate, on the validation set
- The classification error rate, on the test set

Provide answers to these questions:

(b) (5 points) You should find that the error rates on the training sets generally decrease as the number of clusters increases. Explain why.

(c) (5 points) Examine the error rate curve for the test set and discuss its properties. Explain the trends that you observe.

5 Write up

Hand in answers to all the questions in the parts above. The goal of your write-up is to document the experiments you have done and your main findings. So be sure to explain the results. The answers to your questions should be in pdf form and turned in along with your code. Package your code and a copy of the write-up pdf document into a zip or tar.gz file called A2-*your-student-id*.[zip—tar.gz]. Only include functions and scripts that you modified. Submit this file on MarkUs. Do not turn in a hard copy of the write-up.